User Profile Management and Selection in Context-Aware Service Platforms for Networks Beyond 3G

-

A Practical Application of Semantic Web Technologies

Dissertation
zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften (Dr. rer. nat.)
im Fachbereich Elektrotechnik / Informatik
der Universität Kassel

von

Michael Sutterer

Kassel 2009

Tag der Disputation: 1. Oktober 2009

Abstract

In recent years, progress in the area of mobile telecommunications has changed our way of life, in the private as well as the business domain. Mobile and wireless networks have ever increasing bit rates, mobile network operators provide more and more services, and at the same time costs for the usage of mobile services and bit rates are decreasing. However, mobile services today still lack functions that seamlessly integrate into users' everyday life. That is, service attributes such as context-awareness and personalisation are often either proprietary, limited or not available at all. In order to overcome this deficiency, telecommunications companies are heavily engaged in the research and development of service platforms for networks beyond 3G for the provisioning of innovative mobile services. These service platforms are to support such service attributes.

Service platforms are to provide basic service-independent functions such as billing, identity management, context management, user profile management, etc. Instead of developing own solutions, developers of end-user services such as innovative messaging services or location-based services can utilise the platform-side functions for their own purposes. In doing so, the platform-side support for such functions takes away complexity, development time and development costs from service developers.

Context-awareness and personalisation are two of the most important aspects of service platforms in telecommunications environments. The combination of context-awareness and personalisation features can also be described as situation-dependent personalisation of services. The support for this feature requires several processing steps. The focus of this doctoral thesis is on the processing step, in which the user's current context is matched against situation-dependent user preferences to find the matching user preferences for the current user's situation. However, to achieve this, a user profile management system and corresponding functionality is required. These parts are also covered by this thesis. Altogether, this thesis provides the following contributions:

The first part of the contribution is mainly architecture-oriented. First and foremost, we provide a user profile management system that addresses the specific requirements of service platforms in telecommunications environments. In particular, the user profile management system has to deal with situation-specific user preferences and with user information for various services. In order to structure the user information, we also propose a user profile structure and the corresponding user profile ontology as part of an ontology infrastructure in a service platform.

The second part of the contribution is the selection mechanism for finding matching situation-dependent user preferences for the personalisation of services. This functionality is provided as a sub-module of the user profile management system. Contrary to existing solutions, our selection mechanism is based on ontology reasoning. This mechanism is evaluated in terms of runtime performance and in terms of supported functionality compared to other approaches. The results of the evaluation show the benefits and the drawbacks of ontology modelling and ontology reasoning in practical applications.

Acknowledgement

First of all, I would like to thank my supervisor, Prof. Dr. Klaus David, head of the chair for Communication Technology at the University of Kassel, for the opportunity of doing this dissertation. I am very grateful to him for countless discussions and invaluable feedback.

I would also like to express my sincerest appreciation to Dr. Josip Zoric, Senior Research Scientist at Telenor Research and Innovation, and Associated Professor II at the Norwegian University of Science and Technology, for very useful technical discussions and feedback for my doctoral thesis. Many thanks also to Prof. Dr. Kurt Geihs and Prof. Dr. Albert Zündorf as members of the doctoral committee.

Special thanks to Dr. Olaf Drögehorn, who has been a continual inspiration, with whom I had very profitable discussions and who has provided indispensable support and feedback. Thanks as well to all my colleagues at the chair for Communication Technology. We had many interesting and fruitful discussions on countless topics in the area of context-awareness, personalisation and ubiquitous computing.

Furthermore, I would also like to thank all my colleagues from countless companies, universities and research institutes throughout Europe, with whom I have worked in the EU-FP6 projects MobiLife and SPICE. The work in these two projects provided the background and the inspiration for my dissertation.

Finally, I would also like to thank my family and my friends who have supported me throughout the whole time of studying and writing this thesis.

Erklärung

Hiermit versichere ich, dass ich die vorliegende Dissertation selbständig und ohne unerlaubte Hilfe angefertigt und andere als die in der Dissertation angegebenen Hilfsmittel nicht benutzt habe. Alle Stellen, die wörtlich oder sinngemäß aus veröffentlichten oder unveröffentlichten Schriften entnommen sind, habe ich als solche kenntlich gemacht. Kein Teil dieser Arbeit ist in einem anderen Promotions- oder Habilitationsverfahren verwendet worden.

Michael Sutterer Kassel, November 2009

Table of Contents

1 Introduction

In recent years, progress in the area of mobile telecommunications has changed our way of life, in the private as well as the business domain. Mobile and wireless networks have ever increasing bit rates, mobile network operators provide more and more services, and at the same time costs for the usage of mobile services and bit rates are decreasing. However, mobile services today still lack functions that seamlessly integrate into users' everyday life. That is, service attributes such as context-awareness and personalisation are often either proprietary, limited or not available at all. In order to overcome this deficiency, telecommunications companies are heavily engaged in the research and development of service platforms for networks beyond 3G for the provisioning of innovative mobile services. These service platforms are to support such service attributes.

Service platforms are to provide basic service-independent functions such as billing, identity management, context management, user profile management, etc. Instead of developing own solutions, developers of end-user services such as innovative messaging services or location-based services can utilise the platform-side functions for their own purposes. In doing so, the platform-side support for such functions takes away complexity, development time and development costs from service developers.

The service platform model referred to and described in this thesis is only a simplification of real-world service platforms. In reality, such service platforms are much more complex, including numerous layers and enablers, which are not addressed here. Hence, the service platform model described in this thesis should be seen as an approximation of a real-world service platform for the intended domain, in which the description is restricted to that parts that are of interest in this thesis. In addition, the mentioned service platform is directed towards networks beyond 3G. This is because current 3G service platforms or infrastructures do not yet provide advanced context-awareness and personalisation functions in the intended way.

1.1 Motivation

Context-awareness and personalisation are two of the most important aspects of service platforms in telecommunications environments. The combination of context-awareness and personalisation features can also be described as situation-dependent personalisation of services. The support for this feature requires several processing steps:

1. Data about the user's environment has to be captured by sensors
2. Sensor data has to be transformed into meaningful context information
3. The resulting context information has to be matched against situation-dependent user preferences to find the matching user preferences for the current user's context
4. The matching user preferences can be applied for service personalisation

The focus of this doctoral thesis is on the third step, the automatic selection of matching situation-dependent user preferences to support the situation-dependent personalisation of services. However, to achieve this, a user profile management system and corresponding functionality are required. These parts are also covered by this thesis.

One of the basic requirements for this step is to research and develop a user profile management system that manages multiple user data. In particular, it should manage user data required by platform services, e.g. for accounting and billing, and also user data required by end-user services such as innovative messaging services or location-based services. On the one hand, there are general user data such as user name, date of birth, address, credit card details, etc. which may be required by all or many services. Furthermore, there are service-specific user data and preferences such as ring tone preferences for telephony, which is only required by a particular service. Last but not least, there are situation-dependent user data and preferences. For example, a user could have different preferences for the notification of incoming news messages. While the user is at home, she may want to be notified of incoming messages immediately, whereas in a business meeting, she may want the notification to be postponed until after the meeting.

Another important aspect to be considered is the use of different or extensible vocabularies for expressing user attributes. An overall vocabulary that is used on platform side may simply not be sufficient for developers of end-user services. These may require or prefer different vocabularies or may have to extend a vocabulary that is provided by the service platform operator. Extensibility of vocabularies is especially important, because it is not possible to create a common overall vocabulary for user attributes during design time of the service platform. This is so because existing services can be changed, substituted or extended with additional functionality or new services can be added. Still another reason for addressing this issue is the matter of roaming, in which the user enters the network of a foreign network operator and is not directly connected with her home service platform. As the user wants to continue the use of her subscribed services, exchange of user attributes and interoperability of user attribute vocabularies is needed.

Yet another challenge is the process of situation-dependent service personalisation. This process should be carried out automatically without the need for user interaction. Thus, the user should not be the one who has to change the user profile to match her changed situation. Instead, the service platform functionality, i.e. the user profile management system, should control this automatically. In order to achieve this, the user profile management system requires notifications about changes in the user’s situation, selects the matching user preferences and subsequently informs subscribed services about the changed user preferences.

What has not yet been covered above is one of the most important aspects, the user. She is the one who decides about the success and acceptability of personalised services, and hence the success of service platforms. For this reason, services must provide convincing and precise mechanisms for situation-dependent personalisation. At the same time, the user should always have the control of her personal data. The user should know about all stored data that is related to her person, she should be able to view and edit her personal data, and she should be able to activate or deactivate processing of personal data such as location tracking. This aspect also has to be taken into consideration with legal issues, i.e. national laws as well as European directives. Eventually, this means that it would be very beneficial to find an easily understandable way for the normal non-technical user to let her participate in the specification of user data and in particular of situation-dependent user preferences. The challenge here is twofold: on the one hand, situation-dependent user preferences have to be encoded in a machine processible way in order for the user profile management system to provide automatic selection of matching situation-dependent user preferences, and hence to support automatic service personalisation. On the other hand, the user should not be challenged with complex visualisations of user preferences. That is, situation-dependent user preferences should be editable in an easily understandable way.

1.2 Problem Statement

As described above, the field of user profile management in service platforms for networks beyond 3G includes a lot of challenges. Although we will not address all challenges in detail in this doctoral thesis, we will address many of the above aspects in order to reach our main goal. This is to research and develop a user profile and preferences selection module as part of the user profile management system.

Existing research on user profile management systems and related user profile structures does not fully cover the requirements for the targeted service platform. Either they do not provide a means for managing situation-dependent user preferences, or they do not support service specific user data, or they do not cover required functions such as automatic selection of matching user preferences for the current user's situation, or existing solutions do not fit the envisioned requirements in the proper way.

Also, when it comes to how the user can interact with the system in order to specify expressive situation-dependent user preferences, we think that most existing research approaches are not very user-friendly. Some approaches, especially approaches that address learning by evaluating user history and usage data, do not even consider interfacing with the user. And for those approaches, which support the user-driven specification of situation-dependent user preferences, only simple rules are supported. This in turn means that the mechanism for automatically detecting a user profile or preference change cannot be very sophisticated. Finally, user acceptance of such an approach for given reasons will not be very high.

In this thesis therefore, we will investigate how we can improve automatic selection of matching situation-dependent user preferences and profiles in the targeted service platform environment combined with an easily understandable way to define expressive situation-dependent user preferences by the normal non-technical user. In doing so, the user should not be confronted with the complexity of the underlying user profile and user preferences selection mechanism and technology.

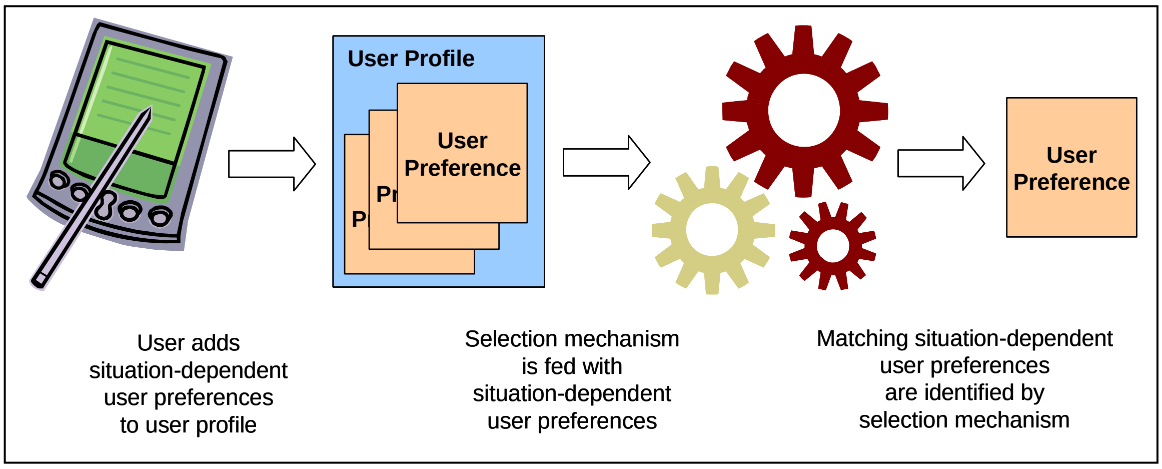

User Profile
User Preference
Pr
P
User Preference
User adds situation-dependent user preferences to user profile
Selection mechanism is fed with situation-dependent user preferences
Matching situation-dependent user preferences are identified by selection mechanism

Figure 1: Processing Steps for User Profile Selection

Figure 1 shows the basic processing steps required for the user profile and user preferences selection mechanism. First, the user enters situation-dependent user preferences to the user profile. Afterwards, the selection mechanism is fed with these situation-dependent user preferences, and identifies the ones that match the current user's situation.

1.3 Requirements and Goals

Based on the descriptions in the previous two subsections, the requirements and goals of this thesis are stated as follows:

Requirement 1:

- The user profile management service in service platform environments should be flexible enough to manage different types of user data

Goal 1:

- Design and implementation of a user profile management system that is capable of managing the following kinds of user data
    a. User data required by platform services such as accounting and billing services
    b. User data required by end user services such as innovative messaging services and location-based services
    c. Service-specific user data that is only required by specific services such as ring tone preferences for telephony services
    d. Situation-dependent user data such as notification preferences for incoming new messages that are related to different situations

Requirement 2:

- The user profile management system should be flexible enough to manage user data that adheres to different or extensible vocabularies for expressing user attributes

Goal 2:

- Design and implementation of a user profile management system that is capable of managing user data that adheres to different and extensible vocabularies for expressing user attributes
    a. It should be possible to manage user data that adheres to different user attribute vocabularies
    b. It should be possible to manage user data that adheres to extensions of user attribute vocabularies

Requirement 3:

- The user profile management system should provide a means to support automatic service personalisation for various services

Goal 3:

- Design and implementation of a user profile management system that supports automatic service personalisation
    a. Design and implementation of a user profile selection module as sub-module of the user profile management system that enables automatic selection of matching situation-dependent user preferences
    b. Design and implementation of a module to receive and process context parameters as input to the user profile selection module

Requirement 4:

- The user profile management system should provide an easily understandable way for the normal user to manage her user data with the focus on situation-dependent user data

Goal 4:

- Design and implementation of a user profile management system that supports an easily understandable way for the normal user to manage her situation-dependent user data
    a. It should be possible for the user to easily specify and edit situation-dependent user preferences
    b. It should be possible for the user to control user data and personalisation features

1.4 Approach

Our approach to research a user profile selection module with the above described goal comprises several steps:

1. Design of a suitable user profile structure
2. Analytic evaluation of search tasks to show the benefits of our user profile structure
3. Design and implementation of a suitable user profile management system
4. Modelling of a user profile ontology
5. Modelling of a location ontology
6. Design and implementation of a user profile selection mechanism
7. Evaluation of the user profile selection mechanism in terms of runtime performance and supported functionality

First, we design and implement a user profile management system that addresses specific requirements for the targeted service platform. This user profile management system is capable of managing various user data. In particular, general user data, service-specific user data and preferences, and situation-dependent user data and preferences have to be taken into account. In order to achieve this, we also have to design a suitable user profile structure that is capable of including all this user data. This user profile structure is also defined by means of an ontology language. Second, the user profile management system addresses the challenge of enabling different user attribute vocabularies to express the actual user data.

The user profile management system will be designed and implemented in a modular way, consisting of several exchangeable modules. One of these is the user profile selection module that selects the matching situation-dependent user profile and preferences for a user's current situation. Our approach for a user profile selection mechanism is based on Semantic Web technologies [1] [2]. In particular, the matching mechanism is based on ontology reasoning capabilities. In this regard, we will also model the ontologies that are used for the ontology reasoning-based user profile selection mechanism.

The use of Semantic Web technologies makes sense, as the trend for representing and exchanging user context in service platforms is also based on Semantic Web technologies. Furthermore, the use of Semantic Web technologies is becoming more and more important. This is because developers and researchers are starting to understand the advantages provided by these technologies. Still, practical applications that use Semantic Web technologies such as ontology reasoning are rare.

Finally, we evaluate our ontology reasoning-based user profile selection mechanism in terms of runtime performance and supported functionality. In doing so, we provide an experience report including advantages and disadvantages of using ontology reasoning capabilities.

1.5 Contribution

As a result of our approach, this thesis makes the following contribution:

- A suitable user profile structure
- An analytic evaluation of search tasks with regard to our user profile structure
- A suitable user profile management system
- A user profile ontology
- A location ontology
- A user profile selection mechanism
- An evaluation of our user profile selection mechanism in terms of runtime performance and supported functionality
- An experience report on using ontology reasoning capabilities

First, in the field of user profile structures and vocabularies, we provide and evaluate a user profile structure and ontology that enables the definition of service- and situation-dependent user preferences. The corresponding ontology is extensible with arbitrary user attribute vocabularies, and user profile exchange and interoperability is supported in the best possible way. Second, in the field of user profile management systems in service platforms we provide a modular system, in which several modules are easily exchangeable.

In addition we contribute to the field of automatic service personalisation, which is an important aspect for the user acceptance and success of future context-aware services. This is done in researching and developing a user profile selection mechanism that takes advantage of ontology reasoning. Also linked to this mechanism is the modelling of ontologies and the evaluation of ontology reasoning performance in terms of execution time and supported functionality related to different variables that influence the ontology reasoning process.

Considering this aspect, the results of our approach can be used as instruction on how to model ontologies, how to use ontology reasoning capabilities, and to understand the advantages and disadvantages of using ontology reasoning capabilities. Furthermore, it can also be used as experiences report. In doing so, service developers can estimate the system properties in the sense of execution time with regard to used software environment, hardware environment, size of database and other parameters of their systems. This can be done based on the measurement results carried out in this thesis. With the resulting estimation, they can then plan their system accordingly beforehand.

Not addressed in this thesis are some very important aspects related to our approach: Firstly, we do not address the issue of user modeling, i.e. we do not learn user models. Secondly, privacy and security issues related to personal data are beyond the scope of this thesis. Thirdly, we do not deal with probabilities related to user context. This means that the user context provided to the user profile management system is supposed to be accurate. Fourthly, we do not research or extend ontology reasoning algorithms. Instead we use several existing software libraries for our purposes. Finally, we do not carry out user evaluations. However, all these aspects need to be addressed in future work in order to further progress made.

1.6 Structure of the Thesis

In chapter 2, we first depict the fundamentals of user profiles and user profile management in service platforms for networks beyond 3G. We then list related work and discuss it with regard to the requirements for service platforms. In chapter 3, we describe our approach of a user profile management system, including the related user profile structure. Chapter 4 is dedicated to ontologies. On the one hand we present the ontology definition for the user profile structure. On the other hand we explain the modelling of a context ontology that is strongly connected to the specification of situation-dependent user preferences. Subsequently, chapter 5 includes the user profile selection mechanism. In chapter 6 we evaluate the user profile selection mechanism in terms of supported functionality and execution time. That is, the supported functionality is compared to that of other approaches that do not include ontology reasoning. As well execution time is compared between different reasoning libraries. In addition, the execution time is also compared between other parameters related to ontology processing. Finally, we conclude in chapter 7 with a summary and discussion of our results and an outlook for future research.

1.7 Publications and Book Chapters

The following list shows the publications and book chapters I wrote during the work on my thesis. The corresponding conferences and workshops such as VTC 2007-Spring, SAINT 2007 and Middleware 2007, the international research and development projects EU-FP6 MobiLife and EU-FP6 SPICE, in which I was involved with big companies, and my work on European and national project acquisition provided me a broad platform for discussion and feedback for the topics addressed in my thesis.

1. M. Sutterer, O. Droegehorn, and K. David, "User Profile Selection by Means of Ontology Reasoning", In proceedings of the Fourth Advanced International Conference on Telecommunications (AICT 2008), IEEE Computer Society Press, ISBN 978-0-7695-3162-5, pp 299-304, Athens, Greece, June 2008.

2. M. Sutterer, O. Droegehorn, and K. David, "UPOS: User Profile Ontology with Situation-Dependent Preferences Support", In proceedings of the First International Conference on Advances in Computer-Human Interaction (ACHI 2008), IEEE Computer Society Press, ISBN 978-0-7695-3086-4, pp 230-235, Sainte Luce, Martinique, February 2008.

3. M. Sutterer, O. Droegehorn, and K. David, "Making a Case for Situation-Dependent User Profiles in Context-Aware Environments", Best Paper Award, In proceedings of the Middleware 2007 Workshop on Middleware for Next-Generation Converged Networks and Applications (MNCNA 2007), ISBN 978-1-59593-932-6, Newport Beach, California, USA, November 2007.

4. D. Bonnefoy, M. Boussard, N. Brgulja, A. Domene, O. Droegehorn, G. Giuliani, R. Kernchen, S.L. Lau, J. Millerat, B. Mrohs, P. Nurmi, P.J. Ollikainen, M. Radziszewski, C. Raeck, M. Salacinski, A. Salden, and M. Sutterer, "Multimodality and Personalisation", chapter 5 in "Enabling Technologies for Mobile Services: The MobiLife Book", John Wiley & Sons, ISBN 0-470-51290-3, pp 153-184, September 2007.

5. P.P. Boda, N. Brgulja, S. Gessler, G. Giuliani, J. Koolwaaij, M. Martin, D. Melpignano, J. Millerat, R. Nani, P. Nurmi, P.J. Ollikainen, P. Polasek, M. Radziszewski, M. Salacinski, G. Schultz, M. Sutterer, D. Trendafilov, and L. Ukropec, "Reference Applications", chapter 7 in "Enabling Technologies for Mobile Services: The MobiLife Book", John Wiley & Sons, ISBN 0-470-51290-3, pp 227-261, September 2007.

6. S.L. Lau, J. Millerat, M. Sutterer, N. Brgulja, O. Coutand, and O. Droegehorn, "Integrating Expert Knowledge into Context Reasoning in Context-Aware Environment", In proceedings of the 7th International Workshop on Applications and Services in Wireless Networks (ASWN 2007), ISBN 978-84-690-5727-8, pp 75-80, Santander, Spain, May 2007.

7. M. Sutterer, O. Droegehorn, and K. David, "User Profile Management on Service Platforms for Ubiquitous Computing Environments", In proceedings of the IEEE 65th Vehicular Technology Conference (VTC 2007-Spring), IEEE Computer Society Press, ISBN 1-4244-0266-2, pp 287-291, Dublin, Ireland, April 2007.

8. M. Sutterer, K. van der Sluijs, O. Coutand, O. Droegehorn, and K. David, "Managing and Delivering Context-Dependent User Preferences in Ubiquitous Computing Environments", In proceedings of the 3rd IEEE SAINT 2007 Workshop on Next Generation Service Platforms for Future Mobile Systems (SPMS 2007), IEEE Computer Society Press, ISBN 0-7695-2757-4, Hiroshima, Japan, January 2007.

9. P. Nurmi, A. Salden, S.L. Lau, J. Suomela, M. Sutterer, J. Millerat, M. Martin, E. Lagerspetz, and R. Poortinga, "A System for Context-Dependent User Modeling", In proceedings of the OTM 2006 Workshop on Context-Aware Mobile Systems (CAMS 2006), Springer, ISBN 978-3-540-48273-4, pp 1894-1903, Montpellier, France, October 2006.

10. A.V. Zhdanova, J. Zoric, M. Marengo, H. van Kranenburg, N. Snoeck, M. Sutterer, C. Raeck, O. Droegehorn, and S. Arbanowski, "Context Acquisition, Representation and Employment in Mobile Service Platforms", In proceedings of the IST Mobile & Wireless Communications Summit 2006 Workshop on Capturing Context and Context Aware Systems and Platforms, Myconos, Greece, June 2006.

11. O. Coutand, M. Sutterer, S.L. Lau, O. Droegehorn, and K. David, "User Profile Management for Personalizing Services in Pervasive Computing", In proceedings of the 6th International Workshop on Applications and Services in Wireless Networks (ASWN 2006), Fraunhofer IRB Verlag, ISBN 3-8167-7111-4, pp 3-11, Berlin, Germany, May 2006.

12. A. Salden, R. Poortinga, M. Bouzid, J. Picault, O. Droegehorn, M. Sutterer, R. Kernchen, C. Raeck, M. Radziszewski, and P. Nurmi, "Contextual Personalization of a Mobile Multimodal Application", In proceedings of the 2005 International Conference on Internet Computing (ICOMP 2005), CSREA Press, ISBN 1-932415-69-6, pp 294-300, Las Vegas, Nevada, USA, June 2005.

My contribution to the works listed above is explained below, in reverse order:

1. In "Contextual Personalization of a Mobile Multimodal Application", an architecture is described that enables the contextual personalisation of a mobile multimodal application. The term contextual personalisation is used to emphasize personalisation means that are based on contextual parameters such as the user's location, time of day, activity, the capabilities of the user's mobile device and other contextual parameters. This personalisation architecture was later refined to support various mobile applications. Besides working on the specification of this architecture and its subcomponents, I implemented the corresponding user profile management as part of this architecture that enables the definition of situation-dependent user preferences. The matching situation-dependent user preferences are selected by different kinds of rule engines, which match the conditions of the user preferences with contextual parameters. This work was part of the EU-IST project MobiLife (Mobile Life), in which I worked as the Task Leader of task 2.3 on Mobile Lifestyle Personalisation Components Research and Development.

2. In “User Profile Management for Personalizing Services in Pervasive Computing”, the main focus is on the challenges and requirements for a user profile management in pervasive computing. In particular it focuses on the challenge of serving various applications with the same base set of user data. The suggested approach was the use of application-specific views on that base set of user data, so that applications can use different vocabularies for describing and using user data. This idea served as one starting point for the finally implemented and evaluated user profile management system of this doctoral thesis.

3. In “Contextual Acquisition, Representation and Employment in Mobile Service Platforms”, ideas were presented on how a mobile service platform can utilize context information from heterogeneous context sources. As part of this work I presented the ideas how to utilize contextual information in the area of user profile management for the contextual personalisation of services. This work was done in the scope of the EU-IST project SPICE (Service Platform for Innovative Communication Environment), in which I worked as Task Leader of task 4.1 on Personal Information Management.

4. In “A System for Context-Dependent User Modeling”, the refined personalisation architecture of the EU-IST project MobiLife was shown, which explains the learning, management and application of context-dependent user models for serving various mobile applications. As part of this User Modeling framework, which was implemented and presented in various demos, I was responsible for the subcomponent on the management of user models.

5. In “Managing and Delivering Context-Dependent User Preferences in Ubiquitous Computing”, the user profile management component developed and used in the EU-IST project MobiLife was presented in more detail, showing the main idea of using application-dependent views on user data on the one hand, and using situation-dependent sub-profiles on the other hand.

6. In “User Profile Management on Service Platforms for Ubiquitous Computing Environments”, the user profile management aspects, already mentioned in some of the above papers has been adapted and described in the background of service platforms in the telecommunication domain as researched and developed in the EU-IST SPICE project.

7. In “Integrating Expert Knowledge into Context Reasoning in Context-Aware Environment”, it is shown, how the user profile management system of the EU-IST project MobiLife is used in a system that integrates expert knowledge in the context reasoning step to learn and apply user preferences in a context-sensitive environment.

8. In “Enabling Technologies for Mobile Services: The MobiLife Book”, I was the main contributor on the sub-chapter 5.2 on Contextual Personalisation that summarises the contextual personalisation architecture approach of the EU-IST project MobiLife. I was also main contributor to the sub-chapter 7.6 on Wellness-aware Multimodal Gaming System as an example of an application that was implemented to show the capabilities of the contextual personalisation architecture researched and developed in that project.

9. In “Making a Case for Situation-Dependent User Profiles in Context-Aware Environment”, parts of this doctoral thesis are shown that include my analytic evaluation of clustering use data into situation-dependent sub-profiles evaluated for two main use cases. More details are shown in section 3.1.2 of this doctoral thesis.

10. In “UPOS: User Profile Ontology with Situation-Dependent Preferences Support”, also parts of this doctoral thesis are shown. In particular, I presented my user profile ontology (UPOS) that is also described in this doctoral thesis. More details are provided in section 4.2.

11. In “User Profile Selection by Means of Ontology Reasoning”, details of my selection mechanism are provided, which explain the ontology reasoning based approach to select matching situation-dependent user preferences used in this doctoral thesis. More details are provided in chapter 5.

2 Fundamentals and Related Work

In this chapter, we first provide the fundamentals related to the thesis. This includes definitions of central terms, fundamentals about service platforms for networks beyond 3G, related user profile management and fundamentals about the Semantic Web. Afterwards related work is shown and discussed.

2.1 Fundamentals

This subsection provides the fundamentals for this thesis. We first introduce definitions for user context, user profile and further terms required for the understanding of the subsequent sections. Second, we present basic knowledge about service platforms, in which our user profile selection mechanism is integrated. Third, user profile management in service platforms is depicted. Finally, we introduce the field of the Semantic Web, of which we use several technologies for our user profile selection approach.

2.1.1 Definitions

For user context, we take the definition from Dey [3], which is as follows:

Context: Context is any information that can be used to characterise the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and application themselves.

For user profiles, user profile subsets and user sub-profiles as used in this thesis, we use the following definitions:

User profile: A user profile is the collection of all user data about a particular user apart from context. This includes general user information such as user name and user address, service-specific user data, situation-dependent user data, user history and data that is inferred from other user data. Not included in a user profile is dynamically changing context such as user location and user activity. Context is only included in the user profile if it is part of situation-dependent user data. In this case, the context is used to describe one or more conditions, i.e. the situation, in which situation-dependent user data is relevant. These conditions are also called situational conditions in the following.

User profile subset: A user profile subset is a subset of all user data of a user profile.

User sub-profile: Synonym for user profile subset.

In addition, also the following term is frequently used in this thesis:

Situational condition: A situational condition is the condition part of a situation-dependent user preference. It describes the situation, in which the situation-dependent user preference is relevant.

2.1.2 Service Platforms for Networks Beyond 3G

Service platforms in telecommunications environments aim to support easy and quick creation, test and deployment of mobile communication and information services. In order to achieve this, service platforms first have to address the design and development of efficient and innovative service creation and execution platforms for mobile services. Key objectives for service platforms are the seamless delivery of mobile services over heterogeneous execution platforms, networks and terminals, offering a personalised user experience anytime and anywhere, simple use of services and devices through context-awareness, personalisation and customisation provided by a trusted platform, and the enabling of service provision over different countries, network operators and service platform operators.

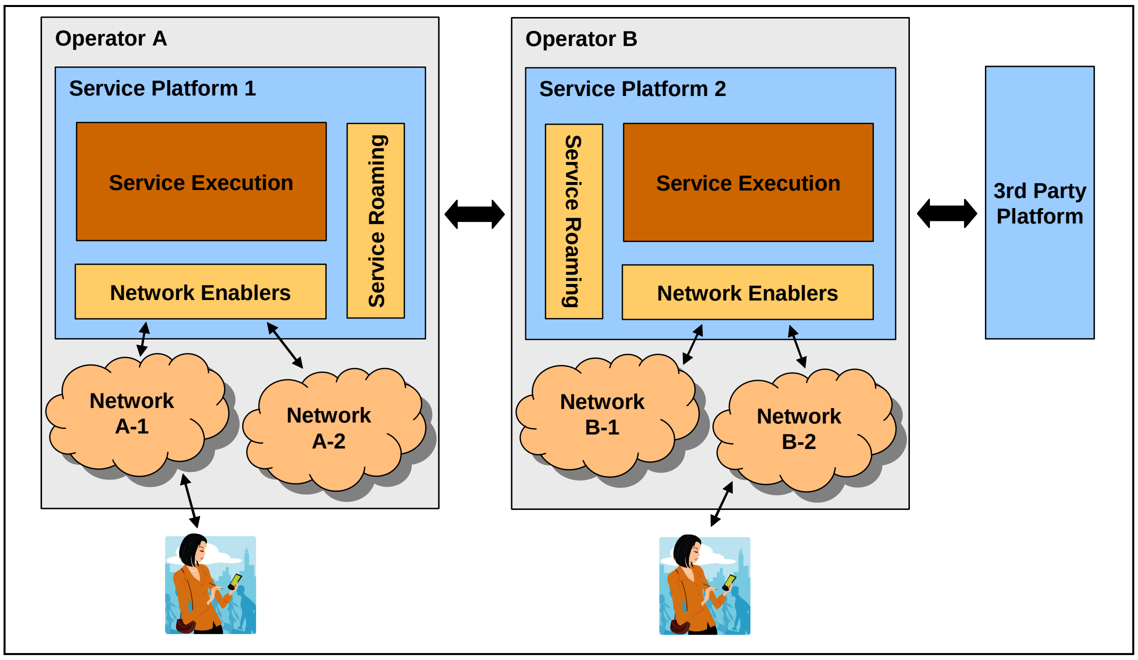


Figure 2: Service Platforms for Networks Beyond 3G

Figure 2 provides a simplified view of service platforms. As mentioned in the introduction chapter, in reality, such service platforms are much more complex architectures, which we will not describe here in detail as many parts of such service platforms are not relevant for this thesis. However, for a more detailed view on an example service platform, see Appendix A: SPICE Service Platform, in which the service platform developed in the SPICE project [4] [5] [6] is shown.

In addition, the shown service platform is directed towards networks beyond 3G. This is because current 3G service platforms or infrastructures do not yet provide the mentioned functions in the intended way.

The entities in Figure 2 are defined as follows:

Operator: The operator of a service platform

Service Platform: A set of components that provide an architecture for service delivery

Service Execution: The environment, in which the services provided by the service platform are executed

Network Enablers: Components that provide access to network capabilities

Network: Communication system that interconnects computer systems or devices at different sites

User: The end user consuming services provided by the service platform.

Service Roaming: The provision of a service in a location that is different from the location where the service was registered

3rd Party Platform: Service platform of third party service provider accessing and providing service components

Based on Figure 2, a user could have several terminals such as a mobile phone, a laptop and a home personal computer with which she wants to access multiple services. Depending on the device and the user's situation, different networks should be available to communicate with the service platform for what the service platform has to provide the corresponding network enablers. In case the user enters the network of a foreign network operator and is not directly connected with her home service platform her services should still be available. This could be achieved by roaming her services from the home service platform to the visited service platform or providing other means to exchange relevant information between the home and visited service platform to support the use of her services. Finally, services could also be provided to users by third-party service providers via other service platforms.

The design of such a service platform has to cover many aspects. These are rapid service introduction to the market, a rich set of service enabler components, compelling user experience, and access control and identity management. Rapid service introduction is addressed with a service creation environment, semantically annotated service descriptions and interfaces, and automatic service composition. Interaction of individual enabler components is addressed with service discovery facilities, service broker functionality and service roaming management.

Compelling user experiences is another important issue as the success of a service platform depends on the user's experiences in terms of user-service interaction and a user tailored service experience. This includes ease of use of services, support for content management and delivery, and context-awareness and personalisation features. The latter one includes the management and selection of situation-dependent user preferences and profiles respectively. More details on the objectives, design and development of service platforms can be found in [4] [5] [6]. Also [7] provides detailed guidelines and experiences in the development of mobile service architectures.

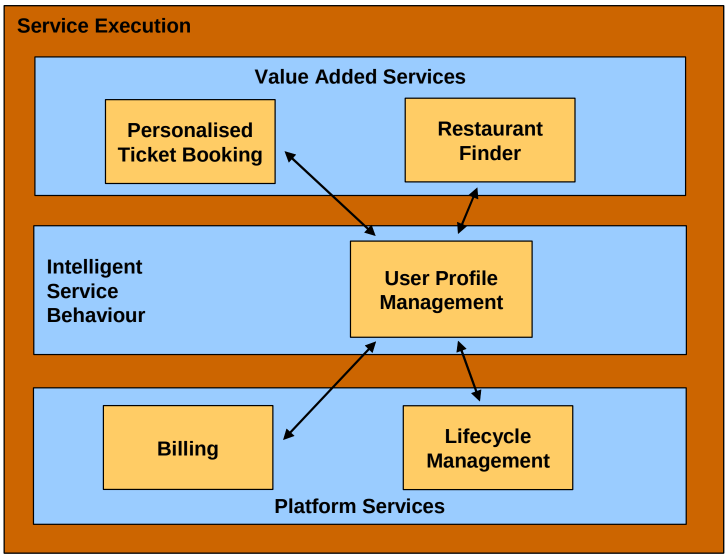


Figure 3: User Profile Management in Service Platforms

Figure 3 provides a simplified view of the service execution environment of service platforms and the classification and interconnection of the user profile management to other types of services. The entities in Figure 3 are defined as follows:

Service Execution: The environment, in which the services provided by the service platform are executed

Platform Services: Base services of the service platform that are needed to manage and operate the service platform

Intelligent Service Behaviour: Services that support Value Added Services in providing intelligent service behaviour such as situation-dependent behaviour

Value Added Services: Services that interact with the end user, also called end-user services or applications

As shown in Figure 3, the user profile management system is classified as service that enables intelligent service behaviour. On the one hand, the user profile management should be capable of managing user data needed for platform services such as billing or lifecycle management. On the other hand, the user profile management should be capable of managing user data for Value Added Services such as a Restaurant Finder or a Personalised Ticket Booking Service, also called end-user services or applications in the following.

2.1.3 User Profile Management

User profile management systems in service platforms for networks beyond 3G differ essentially from ones in stand-alone applications. In order to depict the differences between these two systems, we consider four architectural components, in particular an application core, a user profile management system, a user context management system and a user profile selection module. These are described as follows:

Application Core: The central component of an application that manages the application logic and applies situation-dependent user preferences for the customisation of the offered services

User Profile Management: The component that provides a means to manage user profiles, i.e. to create, set, update, query and delete them

User Context Management: The component that gathers user context, derives high-level descriptions of user context and provides it to interested clients

User Profile Selection Module: The component that evaluates whether user profiles or single user preferences match the user's current situation

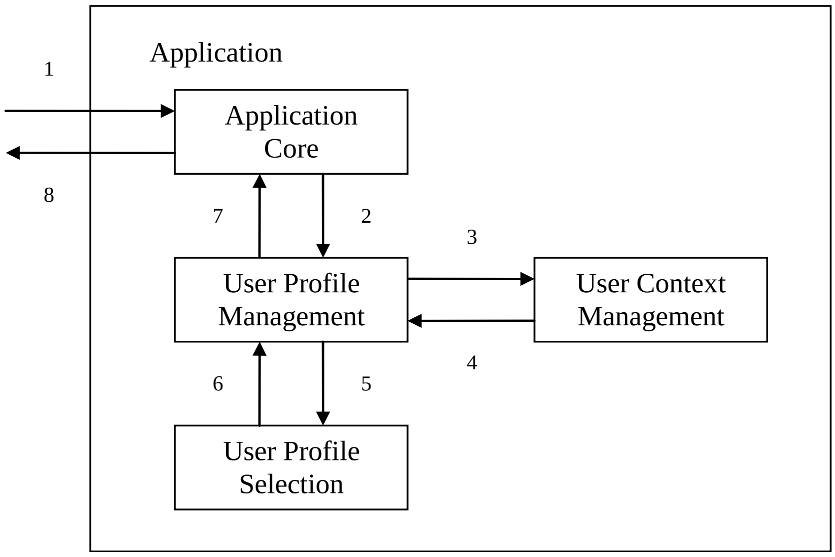


Figure 4: Request / Response Pattern for Stand-Alone Applications

Figure 4 shows how the request / response pattern for user profile query in stand-alone applications could look. In this case, all four architectural components are part of the stand-alone application and hence are application-specific. The sequence steps are as follows:

1. In step 1, a user interfaces with the application core.
2. The application requests the user's preferences from the user profile management subsystem in step 2, in order to execute the user's query.
3. In step 3, the current user's context is requested from the user context management system.
4. The answer in step 4 is returned to the user profile management subsystem
5. In step 5, this answer is passed on to the user profile selection module
6. The result of the user profile selection process is returned to the user profile management system in step 6.
7. Subsequently, the application core receives the user preferences to be applied for the current user request in step 7
8. Finally, the customised service is provided to the user is step 8.

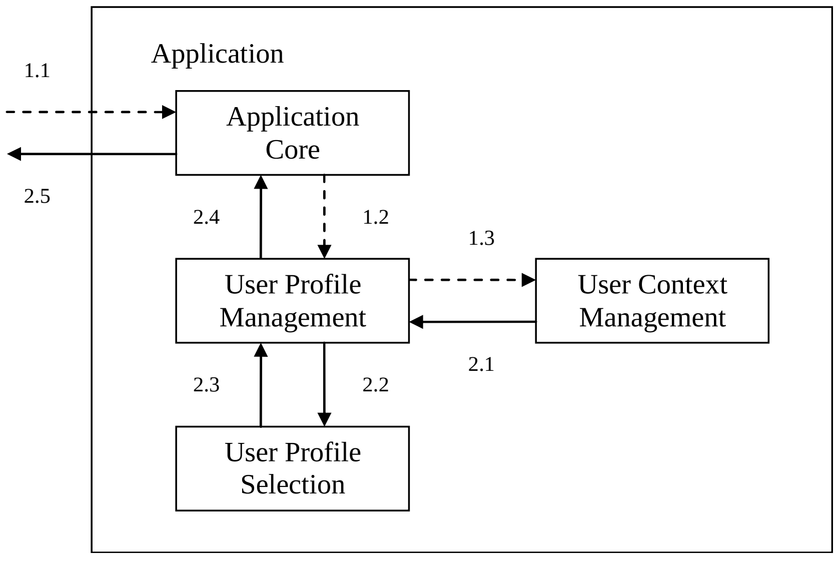


Figure 5: Subscription / Notification Pattern for Stand-Alone Applications

Figure 5 shows the subscription / notification pattern for the same case. The sequence of messages is exactly the same. The only difference is that the sequences are separated into two parts as described below:

1. The subscription steps 1.1, 1.2 and 1.3 are done beforehand so that there is no need for the user to permanently interact with the application. In contrast to this, the user is notified about changes to the subscribed items.
2. The notification process starts with a notification by the user context management system in step 2.1, which is about a change in the user context subscribed to. Subsequently steps 2.2 to 2.5 are then carried out. Step 2.5 does not necessarily have to be reached, as a change in the user's context may not necessarily result in changing user preferences.

An application example in which the subscription / notification pattern makes sense is a news delivery service. By means of subscription, the user can be informed about incoming news as soon as a new message about a particular news item is available. However, whereas the user may want to be notified immediately of incoming news when she is at home, she may want the incoming news to be postponed until after a meeting, in case she happens to be in a business meeting. The approach depicted in Figure 5 aims to support this scenario.

In service platforms, user profile management systems are not application-specific anymore. Besides the user profile management system, the user context management system and user profile selection module could be also placed on service platform side. This makes sense for several reasons. Firstly, application developers do not have to design and implement own modules for user context management, user profile management and user profile selection. This usually significantly decreases application complexity, especially should complex reasoning mechanisms be used for context inference and user profile selection. As a result, development time and development costs for applications can also be decreased. Secondly, there may be a lot of user data that is not application-specific and hence could be shared by many applications. If this user data was managed by each application independently, the same user data would be requested from the user, managed by the application, and would have to be updated by the user in case of modifications for each single application. In case the user profile management system is placed on service platform side, this system could support sharing user data by many applications.

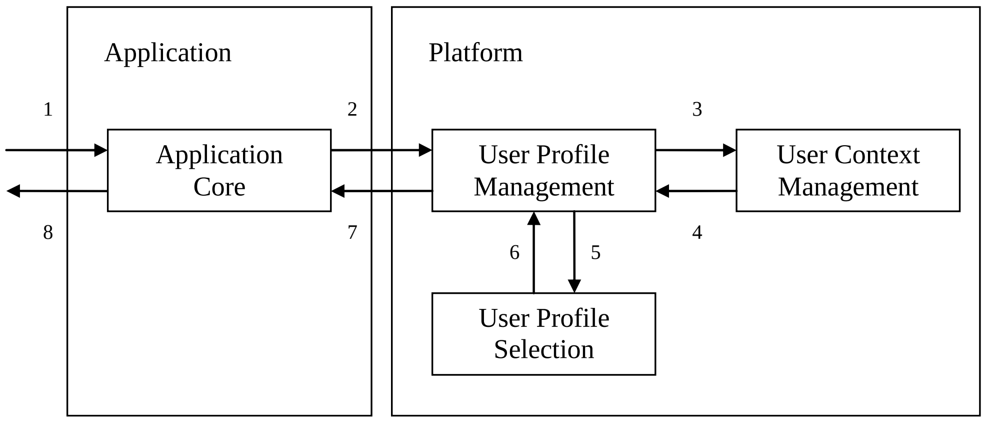


Figure 6: Request / Response Pattern for Applications in Service Platforms

Figure 6 and Figure 7 show the corresponding request / response pattern and subscription / notification pattern for applications that use service platform side user profile management. The single sequence steps for Figure 6 are as follows:

1. In step 1, a user interfaces with the application core.
2. The application requests the user’s preferences from the user profile management in step 2, in order to execute the user’s query. In this case, the user profile management is located within the service platform.
3. In step 3, the current user’s context is requested from the user context management system.
4. The answer in step 4 is returned to the user profile management subsystem
5. In step 5, this answer is passed on to the user profile selection module
6. The result of the user profile selection process is returned to the user profile management system in step 6.
7. Subsequently, the application core receives the user preferences to be applied for the current user request in step 7
8. Finally, the customised service is provided to the user is step 8.

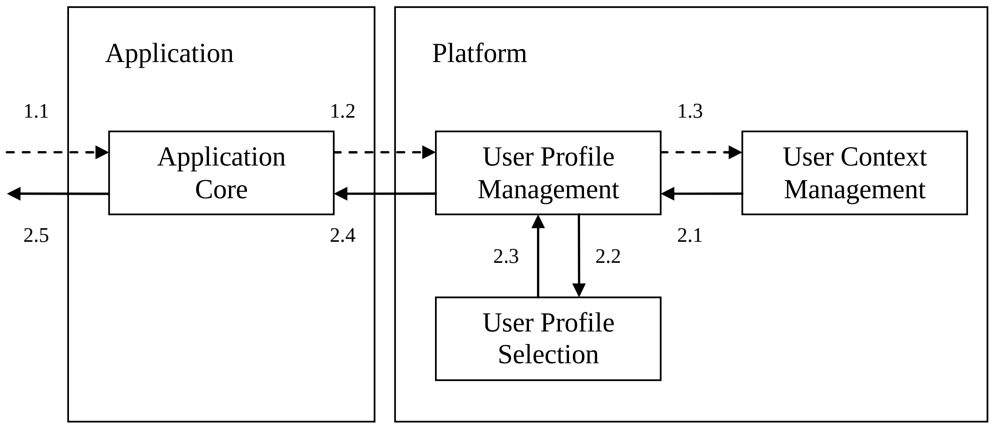


Figure 7: Subscription / Notification Pattern for Applications in Service Platforms

The sequence of messages in Figure 7 is exactly the same as in Figure 6. The only difference is that the sequences are separated into two parts as described below:

1. The subscription steps 1.1, 1.2 and 1.3 are done beforehand so that there is no need for the user to permanently interact with the application. In contrast to this, the user is notified about changes to the subscribed items.
2. The notification process starts with a notification by the user context management system in step 2.1, which is about a change in the user context subscribed to. Subsequently steps 2.2 to 2.5 are then carried out. Step 2.5 does not necessarily have to be reached, as a change in the user's context may not necessarily result in changing user preferences.

The depicted approach for applications in service platforms could also vary. In some cases it would make sense that at least some user context is managed within the application, e.g. should very application-specific context be used that is not, or can not, be supported by the service platform. Also for application-specific usage behaviour and user history it would make sense to manage it in the application as other applications may not be able to process it anyway. In these cases it would then also make sense to place user profile selection, parts thereof or an additional selection or refinement functionality on the application side.

The platform side user profile management poses several requirements. Firstly, a user profile structure and vocabulary is needed, which includes and structures multiple user data to an overall user profile. This includes user data required by platform services, e.g. for accounting and billing, and also user data required by end-user services such as innovative messaging services or location-based services. There are different types of user data to be included:

1. General user data such as user name, date of birth, address, credit card details, etc. which may be required by all or many services
2. Service-specific user data and preferences such as ring tone preferences for telephony, which is only required by a certain service or a certain group of services
3. Situation-dependent user data and preferences, which usually are also service-specific

At the same time, a query language is needed in order to request specific parts of the overall user profiles.

In addition, the interaction between the user profile management and the user context management requires a common context model within the service platform. As the high level user context that is derived by the user context management system has to be processed by the user profile selection mechanism in order to find matching user profiles or parts thereof, a common representation and understanding of context is needed.

In service platforms, many other aspects related to user profile management may be of interest, which are not addressed in this thesis, e.g. distribution and synchronisation of user profiles to mobile devices. However, the main focus in this thesis is on the user profile selection mechanism that is based upon a centralised user profile management system and extensible user profile and context ontologies. In section 2.2, we have a closer look and a discussion on the related work on user profile management systems and related challenges.

2.1.4 Semantic Web Technologies

In 2001, Berners-Lee, Hendler and Lassila presented the so-called Semantic Web [8]. The Semantic Web could be described as an extension of the Web, in which information is given well-defined meaning. Whereas information in the Web so far was targeted mainly at human users, the Semantic Web was thought to make information in the Web usable by machines. This idea was also summarised by means of the Semantic Web Architecture, also called Semantic Web Stack, see Figure 8, which was introduced by Berners-Lee [9].

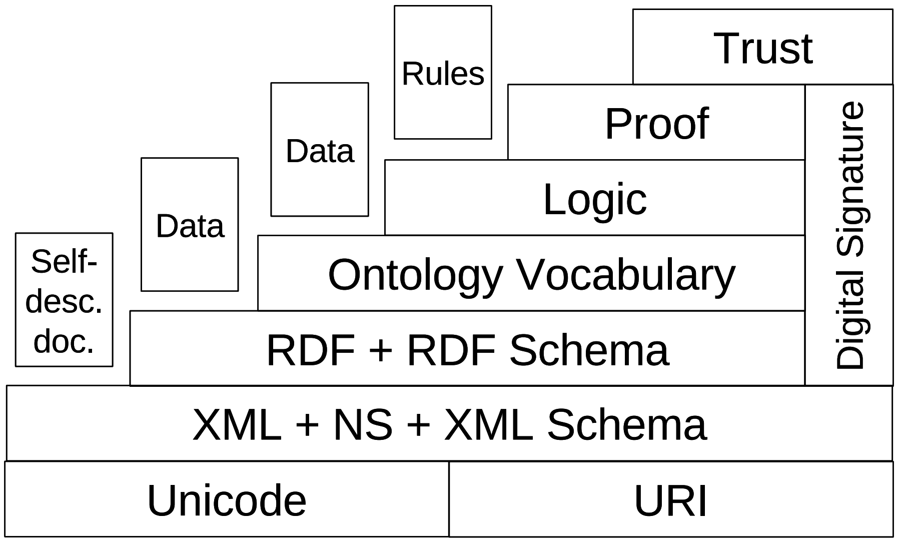


Figure 8: Semantic Web Stack

The base of the Semantic Web Stack is Unicode [10] and URI (Uniform Resource Identifier) [11]. Unicode provides a coding standard for data and an international universal character set that unifies multiple individual character sets such as the Latin, Arabic and Japanese one. URI on the other hand provides a standardised way of identifying Web resources. In [12], this layer is also called Unique Identification Mechanism. The next layer comprises XML (Extensible Markup Language) [13], NS (Namespaces) [14] and XML Schema [15]. XML allows users to add arbitrary structure to documents. However, it does not say anything about what the structure means. Contrary to HTML (Hypertext Markup Language) [16], XML does not mix up the content of a document with the representation of the content. This makes XML much easier to be machine processed than HTML. XML Schema as well enables the definition of a schema, i.e. a vocabulary and structure for the XML document that defines element and attribute names. Finally, NS enables the declaration of namespaces and namespace prefixes. As a result, this layer enables so-called self-describing documents. In [12], this layer is also called Syntax Description Language.

The next higher layer is the RDF (Resource Description Framework) [17] and RDF Schema [18] layer, which could also be called Meta-Data Data Model following the suggestion of [12]. RDF is basically about meta-data. That is, it offers a mechanism to say something about the actual data, which is done by adding meta-data to the actual data. RDF data is represented by an RDF graph in which everything is expressed as triples consisting of a subject, a predicate and an object. RDF Schema on the other hand is RDF's vocabulary description language, which provides a means to define the meaning of RDF data. It can be seen as semantic extension of RDF. The RDF layer is followed by the Ontology Vocabulary layer. An ontology could be defined as an explicit specification of a conceptualisation [19]. Today’s technology, which is used in this layer, is OWL (Web Ontology Language) [20]. This layer even adds additional meta-data on top of RDF Schema with the goal to enable inference and extra functionality.

The Logic layer adds rules to the ontology layer and aims to provide additional inference capabilities. This is the layer currently under development. So far, there is no final technology available. Finally, the Proof layer aims to prove assumptions and the Trust layer aims to add trust mechanisms for validating information. The verification that information has been provided by a specific trusted source could be implemented by means of digital signatures. However, the Proof and Trust layers are currently rather speculative as long as the Logic layer is under development. A detailed discussion about these layers and advanced refined Semantic Web Stacks can be found in [8] [12] [21] [22]. General information on Semantic Web technologies can be found in e.g. [1] [2].

The layers we are interested in are the base Unicode and URI layer up to the Ontology Vocabulary layer. On the one hand, we use Ontology Vocabulary layer technology, i.e. OWL, for specifying a user profile ontology, user attribute vocabularies and context ontologies. On the other hand, we use RDF for the exchange of instances of user profile data and user context. In chapter 3, the use of RDF with regard to user profile and user context instances is described in more detail. In addition, the query process for the selection of matching user profiles and preferences is carried out by means of SPARQL queries [23], the standard RDF query language.

By now, there are a variety of tools for generating and working with RDF and OWL. While e.g. Jena2 [24] [25], KAON2 [26] [27], OWL API [28] and Sesame [29] provide functionality for manipulating RDF and OWL documents, Pellet [30], FaCT++ [31] [32], RacerPro [33] [34], Jena2 [24] [25], KAON2 [26] [27] and others provide ontology reasoning functionality. Some of these libraries are used for our purposes as explained in more detail in subsequent chapters.

2.2 Related Work

In this section, we present exiting work that is related to this thesis. This includes existing work on user profile schemas and ontologies, user profile management systems, user profile selection mechanisms, context-aware systems, and ontologies for user situations.

2.2.1 User Profile Schemas and Ontologies

This subsection lists industry standards and research on user profile schemas and ontologies. As mentioned in section 2.1.3, existing work should support the following types of user profile data:

1. General user data such as user name, date of birth, address, credit card details, etc. which may be required by all or many services
2. Service-specific user data and preferences such as ring tone preferences for telephony, which is only required by a certain service or a certain group of services
3. Situation-dependent user data and preferences, which usually are also service-specific.

The user profile schemas and ontologies are analysed based on their capability to support the following goals as stated in section 1.3 and repeated here:

Goal 1:

- Design and implementation of a user profile management system that is capable of managing the following kinds of user data
    a. User data required by platform services such as accounting and billing services
    b. User data required by end user services such as innovative messaging services and location-based services
    c. Service-specific user data that is only required by specific services such as ring tone preferences for telephony services
    d. Situation-dependent user data such as notification preferences for incoming new messages that are related to different situations

Here, this means that the user profile schema or ontology should provide a user profile structure for the four listed aspects of goal 1.

Goal 2:

- Design and implementation of a user profile management system that is capable of managing user data that adheres to different and extensible vocabularies for expressing user attributes
    a. It should be possible to manage user data that adheres to different user attribute vocabularies
    b. It should be possible to manage user data that adheres to extensions of user attribute vocabularies

Here, goal 2 means that the user profile schema or ontology should enable a user profile structure that can include or be extended with different user attribute vocabularies.

Goal 4:

- Design and implementation of a user profile management system that supports an easily understandable way for the normal user to manage her situation-dependent user data
    a. It should be possible for the user to easily specify and edit situation-dependent user preferences
    b. It should be possible for the user to control user data and personalisation features

Here, goal 4 means that the user profile schema or ontology should provide a means for describing situational conditions in a way that they can easily be visualised to a usual non-technical user to define or edit them.

2.2.1.1 Standards and Specifications

The User Profile Management specification [35] of the European Telecommunications Standards Institute (ETSI) provides guidelines relevant to users' needs to manage their profiles for the personalisation of services and terminals in telecommunications environments. The ETSI Technical Committee Human Factors, which produced the specification, considers effective user profile management as critical to the uptake and success of new and advanced communication services. Hence, it is seen as important to focus on the users' requirements in this area. The user profile concept depicted in this specification addresses different user profile types such as base user profile, device and service user profiles, and situation-dependent user profiles, as well as scenarios, requirements, set-up and maintenance of user profiles, profile activation, and information sharing and privacy. Hence, this work includes very interesting ideas and concepts for our work. However, the document only provides guidelines. This means that it does not provide detailed suggestion on a formal user profile structure, on what technologies to be used, and on how to implement these guidelines.

The 3rd Generation Partnership Project (3GPP) produced several documents for the Generic User Profile (GUP) specification [36] [37]. These specifications address the fact that having several domains within mobile systems (e.g. Circuit-Switched, Packet-Switched, and IP Multimedia Subsystem) and access technologies (e.g. GERAN, UTRAN, and WLAN) introduces a wide distribution of data associated with the user. In order to address the challenge of harmonised usage of the user-related information located in different entities, on the one hand, GUP proposes a reference architecture in specifying GUP functionalities, functional entities, and procedures. On the other hand, GUP proposes a user profile structure by means of a Data Description Method (DDM) and a Datatype Definition Method (DtDM). The proposed user profile structure provides a top-level schema for user profiles, which could consist of several user sub-profiles. However, it does not provide a specification of user attributes, but it shows several examples for individual user attribute schemas that could be used in connection with GUP. The proposed user profile structure is based on XML Schema [15]. Unfortunately, the specification does not provide any specific means to manage situation-dependent user preferences or sub-profiles. This means that this work fulfils goal 1 partially, not including the issue of situation-dependent user data, goal 2 is fulfilled, but goal 4 is not fulfilled for the same reasons as the missing issue in goal 1. Nevertheless, this work provides an interesting approach about user sub-profiles, which we partly took over.

Also the World Wide Web Consortium (W3C) produced a specification, i.e. a W3C recommendation, in the field of user preferences, the Composite Capabilities/Preference Profiles (CC/PP) [38]. CC/PP profiles are descriptions of device capabilities and user preferences. These descriptions are often referred to as a device's delivery context and can be used to guide the adaptation of content presented to that device. The CC/PP specification provides a top-level schema, which could be used for user profiles consisting of different sub-profiles. The CC/PP schema is defined with RDF Schema [15]. The specification also includes a CC/PP attribute vocabulary for print and display as an example for attribute vocabularies. Another example attributes vocabulary is the User Agent Profile (UAProf) specification [39] by the Open Mobile Alliance (OMA). UAProf includes hardware and software characteristics of the device as well as information about the network to which the device is connected, but is distinct from a user preference profile. An extended discussion on CC/PP is also given in [40]. However, CC/PP does not include a concrete user attribute vocabulary. Furthermore, it also does not provide any specific means for the specification of situation-dependent sub-profiles or preferences. This means that this work fulfils goal 1 partially, not including the issue of situation-dependent user data, goal 2 is fulfilled, but goal 4 is not fulfilled for the same reasons as the missing issue in goal 1. Nevertheless, the ideas, which are similar to that of GUP presented above, are partly taken over in this thesis, in particular the approach of an overall user profile consisting of several sub-profiles.

Other well-known vocabularies are the Friend-Of-A-Friend (FOAF) vocabulary [41] and the vCard vocabulary [42]. The FOAF project is based around the use of machine readable Web homepages for people, groups, companies and other kinds of thing. To achieve this, the project provides a collection of basic terms that can be used in these Web pages. The initial focus of FOAF has been on the description of people, since people are the things that link together most of the other kinds of things described in the Web. VCard on the other hand defines a format for an electronic business card. The format aims to be an interchange format between applications or systems and is defined independently of the particular method used to transport it. Both FOAF as well as vCard provide user attributes but no user profile structure for specifying application-specific or situation-specific sub-profiles. As result, they do not cover goal 1, goal 2 and goal 4. As will be mentioned later, both these schemas will be considered as possible user attribute schemas that could be used in combination with a user profile structure for our approach.

2.2.1.2 User Model and User Profile Ontologies

In [43], Jon Orwant describes the Doppelgänger User Modeling System. This system gathers data about users, performs inferences upon the data, and provides resulting information to applications. This functionality is supported by heterogeneous learning techniques that are implemented in an application-independent and sensor-independent environment. An interesting part of this work is the definition of user models, which consist of so-called domain submodels and conditional submodels. Domain submodels contain information about a particular aspect of the user’s behaviour. This could e.g. be location information or his preferences for personalised news-paper content. Conditional submodels on the other hand contain user information that supersedes information of domain submodels when a certain condition exists such as a certain time of day or a particular activity. This approach is interesting for the aims of this thesis as the intended user profile management system has to manage user information of many different applications and for different situations. As a result, domain submodels could be used as application-specific submodels, and conditional submodels could be used for situation-dependent user information. Hence, goal 1 and goal 2 can be met. However, the idea of submodels is only expressed in an abstract way without defining a formal structure, which is why we can take over the idea but not any concrete solutions. Based on this, goal 4 cannot be met, as there is no concrete solution. The approach for our user profile structure will take over this idea of domain submodels (application-specific) and conditional submodels (situation-specific).

In [44], [45] and [46], Dominik Heckmann presents the General User Model Ontology (GUMO). This ontology is used for the uniform interpretation of distributed user models in intelligent Semantic Web enriched environments. For this purpose it aims to be a commonly accepted top-level ontology for user models in order to simplify the exchange of user models between different user-adaptive systems. The ontology includes the user's dimensions that are modelled within user-adaptive systems, i.e. general user data such as the user's age and date of birth, user context such as the user's heart beat and current position, and user interests and preferences such as reading poems and drinking certain French Bordeaux wines. However, this user model ontology could be described as a user attribute vocabulary. That is, it is possible to describe a user by means of a user model, but it does not provide any means to describe a user profile structure where a user profile could consist of several sub-profiles or several user submodels. As a result, it does not cover goal 1, goal 2 and goal 4, but this work is consideres as one possible user attribute vocabulary, which can be used in our approach of a service-specific vocabulary.

In [47], a user profile ontology is shown that aims to be a general, comprehensive and extensible reference model to support personalisation, adaptivity and other user-centric features. This ontology is basically similar as the General User Model Ontology above. It provides a user attribute vocabulary, but no means of a user profile structure that could include several user sub-profiles. Also this one does not cover goal 1, goal 2 and goal 4, but it is also considered as possible user attribute vocabulary for our approach.

2.2.1.3 Discussion

Some of the depicted research work and user profile standards already fulfil the needed requirements partially. GUP [36] as well as CC/PP [38] provide a top-level user profile schema that could include several user sub-profiles. For each of the user sub-profiles a different individual user attribute vocabulary could be used to express the actual user information. This two-step mechanism could be used for specifying application-specific sub-profiles. However, both schemas are not sufficient for our goal to specify situation-dependent user sub-profiles or preferences. Indeed, situation-dependent user sub-profiles or preferences with the corresponding situational conditions could be specified within the individual user attribute vocabulary. However, this would lead to a situation in which the evaluation of situational conditions could only be carried out by the corresponding application, as situational conditions are described in an application-specific format. In contrast to this, our goal is to make the evaluation of situational conditions application-independent, i.e. the evaluation should be carried out by a platform service. In particular, this platform service is the user profile management system or a subsystem thereof. Hence, situational conditions have to be independent of a specific application and therefore part of the top-level user profile schema. This means that these two works fulfil goal 1 partially, not including the issue of situation-dependent user data, goal 2 is fulfilled, but goal 4 is not fulfilled for the same reasons as the missing issue in goal 1. Nevertheless, they do provide a good base for additional extensions or modifications as required in this thesis.

Other schemas such as FOAF [41], vCard [42] and GUMO [44] provide a user attribute vocabulary, but no top-level user profile schema. Thus, they can be used to express the actual user attributes but they do not provide any means in order to assemble different sets of user attributes to an overall user profile consisting of several user sub-profiles. As result, they do not cover goal 1, goal 2 and goal 4. However, assuming that a common top-level user profile schema is available that can be extended with application-specific and situation-specific user sub-profiles, these and other user attribute schemas could be used for the application-specific part of a user sub-profile. Finally, this is the decision of the application developers and not of the service platform or user profile management designers.

Furthermore, in order to easily exchange and share user profile information in service platforms, the specification of the user profile schema and user attribute schemas should be defined with a standardised representation language. As semi-automatic transformation of profile instances should be taken into account to support service platform administrators, a representation language has to be selected that supports this scenario. In this transformation process, a user sub-profile instance that adheres to a particular user attribute schema has to be transformed into a user sub-profile instance that adheres to another particular user attribute schema. This scenario could arise should a service platform user subscribe to a new service, which she has not used before. In this case there is no user information for this new service available. However, the required user information could be translated at least in a semi-automatic way from existing user information available for other services the user has already subscribed to. For this purpose, schema matching algorithms for the creation of mappings between different user attribute vocabularies could be applied [48] [49]. Comprehensive surveys of matching algorithms are presented in [50] [51] [52].

Besides user profile information, a lot of other data is managed and exchanged within a service platform. For example, there are context data, service descriptions, quality of service (QoS) data, presence data and data about contents. As there are correlations between different kinds of data, we should not consider the development of user profile related issues as an independent domain. Instead user profile information should be seen as part of an overall data model, which should enable interoperability, easy exchange, reasoning capabilities and sharing capabilities. The most promising language to be used in order to support these goals and the discussed semi-automatic schema transformation is OWL [20]. Hence, the extension of existing user profile schemas such as GUP [36] seems not advisable, as the GUP schema is specified with XML Schema [15].

Another important issue to be considered during the specification and structuring of user profiles is the interfacing between the user and the user profile management system for editing user data. It has to be ensured that user profile editors make user profiles editable in an easy way. For this purpose, a user-friendly user profile representation is needed. Besides [35], which addresses human factors related to this issue and which has been discussed above, for example Pazzani [53] presents a user study on the representation of electronic mail filtering profiles that addresses this issue. In [54] and [55] requirements for user-friendly user profile management are discussed that are also related to user profile representation. In particular, they discuss how users can be involved in setting up user profiles and improve user profile content that has been learned from user behaviour. As a conclusion, a user-friendly representation and structuring of user profile contents should be taken into consideration at design time of user profiles.

2.2.2 User Profile Management and Context-Aware Systems

This sub-section on the one hand includes related work on user profile management systems, and on the other hand on context-aware systems. It is also closed with a discussion.

The user profile management system and context-awareness systems are analysed based on their capability to support the following goals as stated in section 1.3 and repeated here:

Goal 1:

- Design and implementation of a user profile management system that is capable of managing the following kinds of user data
    a. User data required by platform services such as accounting and billing services
    b. User data required by end user services such as innovative messaging services and location-based services
    c. Service-specific user data that is only required by specific services such as ring tone preferences for telephony services
    d. Situation-dependent user data such as notification preferences for incoming new messages that are related to different situations

Here, this means that the user profile management system should provide a means for managing user profiles and user profile structures that enable the four listed aspects of goal 1.

Goal 2:

- Design and implementation of a user profile management system that is capable of managing user data that adheres to different and extensible vocabularies for expressing user attributes
    a. It should be possible to manage user data that adheres to different user attribute vocabularies
    b. It should be possible to manage user data that adheres to extensions of user attribute vocabularies

Here, this means that the user profile management system should provide a means for managing user profiles that enable the two listed aspects of goal 2.

Goal 3:

- Design and implementation of a user profile management system that supports automatic service personalisation
    a. Design and implementation of a user profile selection module as sub-module of the user profile management system that enables automatic selection of matching situation-dependent user preferences
    b. Design and implementation of a module to receive and process context parameters as input to the user profile selection module

2.2.2.1 User Profile Management and Selection

The Generic User Profile (GUP) specification, which was already introduced in sub-section 2.2.1.1, does not only specify a user profile structure by means of a Data Description Method (DDM), but also a reference architecture [37] by specifying GUP functionalities, functional entities, and procedures. GUP functionalities are e.g. authentication and synchronisation, functional entities are e.g. GUP server and GUP data repository, procedures are e.g. create, subscribe and notify procedures. However, neither the specified functional entities nor the procedures address any means for querying and selecting situation-dependent user data. Furthermore, the functional entities also do not address the translation and mapping between different user attribute schemas. As a result, this work partially covers goal 1 and goal 2, excluding the management of situation-dependent user data. It does not cover goal 3, but can be used as starting point requiring several extensions to match our approach.

In [56], a profile management system for personalised services provisioning is presented. In this system, the focus is on the management of user preferences for presentation and usage of telecommunications services to which a user is subscribed. Furthermore, this system assumes that user data are distributed. In this system, User Profile objects include interface preferences on the one hand, and service preferences on the other hand. Interface preferences are defined as terminal and network dependent user preferences that can include the user's preferred ring tone, colour, message encoding, settings such as language, size of characters, etc. Service preferences on the other hand are defined as user preferences that are specific to subscribed services. A user can have several such user profiles that altogether form a so-called Personal Service Environment (PSE). Such PSEs are defined and explained in the Virtual Home Environment (VHE) concept [57] of the 3rd Generation Partnership Project (3GPP). According to VHE, a PSE is a combination of a list of subscriptions, preferences associated with those services, terminal interface preferences and other information related to the user's experience of the system. Within the PSE, the user can also manage location and temporal preferences. The VHE specification, as well as the approach in [56] that is based on the VHE vision, also aims at static as well as dynamic selection of the appropriate user profile. Whereas in static selection, the user explicitly selects a user profile, dynamic selection functionality aims at automatically selecting the best matching user profile. However, both VHE and [56] do not provide any concrete mechanism or proposal on how such automatic selection functionality could be implemented. Furthermore, they both also do not address the translation and mapping between different user attribute schemas. As a result, these works partially cover goal 1 and goal 2, excluding the management of situation-dependent user data. Goal 3 is not covered. Hence, similar as for the GUP architecture above, these works serve as starting point, to which we will add a concrete automatic selection functionality.

In [58] and [59], van der Sluijs and Houben present the Generic User Model Component (GUC). This component manages application-specific user model schemas and related instances, and aims to support interoperability of user models between different applications. For this purpose, it also provides a means to store schema mappings between different user model schemas. These schema mappings can be used to create instance mappings. Finally, in mapping different application-specific user model instances, the interoperability of user data between different applications can be supported. The required mappings can either be created manually by a system designer or semi-automatically with tool support. Example tools [48] [49] [50] [51] [52] were already mentioned in sub-section 2.2.1.3. An example for a schema mapping is shown in [59], which is based on the Semantic Web Rule Language (SWRL) [60]. SWRL can be seen as an extended OWL [20] that in addition provides support for rules. Hence, on the one hand the GUC approach provides mechanisms for supporting interoperability of user models. On the other hand, it does not focus on situation-dependent user profiles and preferences and any related selection functionality. All in all, this work follows a similar approach as ours. The difference is that our approach will focus on situation-dependent personalisation, whereas this work provides concrete solutions for user data interoperability, i.e. goal 2 is fully met, whereas goal 3 is not focused in this work. Goal 1 is partially met, excluding the management of situation-dependent user data. Hence, this work would be a good candidate to complement the work in this thesis, as it does not only support different user attribute vocabularies, but also provides solutions to map these vocabularies for reuse and interoperability of user data.

Groppe and Mueller [61] present a profile management technology for smart customisations in private home applications. The aim of this work is a profile management framework for situation-dependent customisation in smart home environments. However, the focus is on a textual explanation of profile descriptions and profile computation for the creation and modification of profile instances. The automated environment customisation includes different processing steps. First, the environment is monitored and a soft-context profile is created that represents the state of the environment. Second, the soft-context profile is matched with the user profiles. As a result, a customisation profile is created that is used for the customisation of devices and applications. This matching step also covers preference and device conflicts. This is, this work in principal covers the situation-dependent user preferences aspect of goal 1 and goal 3. However, the presented description does not provide any concrete mechanism or algorithm how the selection of matching user preferences is carried out. Furthermore it also does not cover user profile schemas and user attribute schemas, i.e. goal 2. Nevertheless, this work provides an interesting view on the required processing steps.

In [62], an advanced adaptability and profile management framework for the support of flexible mobile service provision is presented. This framework aims to fulfil requirements towards the situation-aware provision of ubiquitous personalised multimedia services in beyond 3G communication systems. The framework is described by means of class diagrams and interfaces of the adaptation engine and profile representation objects. The proposed mechanism is designed to be generic so that the actual adaptation algorithms can dynamically be loaded at runtime. Furthermore, the proposed design also enables the use of arbitrary semantics of adaptation algorithms and user profile data. However, the framework does not include any concrete adaptation mechanisms, i.e. any algorithms for the selection of matching user profile data, in order to realise the intended vision. Hence, goal 3 is met on a high level by covering this functionality, but without concrete solutions. Goal 2 is not covered, and goal 1 is partly covered, but without showing a detailed user profile structure.

Another user profile management framework for context-aware services is given in [63]. In this framework, static user-related information such as user address book, service IDs, schedule information, and dynamic user-related information such as the current user position and situation, purchase history and usage of nearby computing are collected and can be disseminated to other services. A user profile could be described as consisting of several parts, such as a Location Profile and a Purchased Commodity Profile. In case an application requires user profile data for the situation-dependent personalisation of services, it requests the required user data from the user profile management framework and processes this data. As a result, the actual processing, i.e. the selection of best matching user preferences for the current user situation, does not take place in the user profile management framework. Instead, this processing is application-specific. In contrast to this, we aim to include this selection functionality in an application-independent way into the profile management system placed within the service platform. Hence, this work follows another idea than we do and does not cover goal 3. Besides, this framework also does not include any concrete selection mechanism used by a particular profile management client.

Last but not least, Etter, Dockhorn Costa and Broens [64] present a rule-based approach towards context-aware user notification services. This approach aims to rapidly develop applications that provide context-aware notifications without the need to write programming code to activate rules, nor to implement personalised notifications. Instead, the activation of rules and implementation of personalised notifications is provided by an architecture, the so-called Awareness and Notification System. In this system, a Controller module is responsible for evaluating notification rules that follow the Event-Condition-Action pattern. Rules are expressed in a rule language that has been developed specifically for this system with the aim to enable complex event relations, convenient use for application developers and extensibility. In order to manage the notification rules, the system has a Notification Profile Manager module. However, this system is mainly about management and activation of user notifications and not about general user profile management. That is, it does not address the management of user profiles in general, i.e. goal 1, and the selection of matching user data. It does address the selection and notification of matching user notifications. Furthermore, different user profile schemas and interoperability of user profile information are also not covered, i.e. goal 2. However, as it includes interesting ideas and approaches concerning notification functionality, the ideas will be considered in this thesis, too.

2.2.2.2 Infrastructures for Context-Aware Systems

In [65] and [66], Roman et al. present a middleware infrastructure for active spaces, the so-called Gaia meta-operating system. Gaia is designed to support the development and execution of portable applications for active spaces. Active spaces are defined as programmable ubiquitous computing environments in which users interact with several devices and services simultaneously. Gaia offers the five basic services Event Manager, Context Service, Presence Service, Space Repository and Context File System. The Event Manager distributes events in the active space, for example when a user enters the active space, and implements a communication model based on suppliers, consumers and channels. The Context Service is an infrastructure that lets applications query and register for particular context information used for adapting the application to user behaviours and activities. The Presence Service maintains information about active space resources such as devices and present people, and the Space Repository stores information about all software and hardware entities in the space. Finally, the Context File System uses application-defined properties and environmental context information to simplify tasks such as making personal data automatically available to applications conditioned on user presence, organising data to facilitate locating relevant material for applications and users, etc. Whereas the Gaia architecture serves applications with relevant context information, the customisation of applications based on Gaia provided context information is carried out within the client applications. Hence, user profile management is not addressed within Gaia and is application-specific.

An extension and a more detailed view on Gaia are presented in [67]. This system in addition provides agents a choice of reasoning and learning mechanisms, which they can use to understand and react to context. These mechanisms could e.g. be used to learn context-dependent user preferences. However, these mechanisms are provided in the form of libraries that the agents can use. That is, again the user preferences, user profiles and potential selection mechanism takes place at application side instead of at platform side as intended in this thesis.

Also the authors of [68], [69] and [70] present context-aware systems for pervasive computing environments. The infrastructure in [68] is a proposal for an OSGi-Based infrastructure. The proposed infrastructure aims to be independent of any particular hardware platform. OSGi was formerly known as Open Services Gateway Initiative but nowadays is an absolute name for an open standards organisation. In [69], a high-level programming model for pervasive computing environments is presented, called Olympus. Also the Aura project [70] aims to support environments that adapt to the user's needs. All three projects have in common that they aim to support applications with relevant context information so that applications can adapt to the user's situation. However, they also have in common that the actual user preferences, profiles and related processing are carried out within the applications. Solely the Aura project [70] addresses user intents in the infrastructure, but without giving any details on how they are maintained and processed.

Henricksen et al. [71] [72] [73] [74] [75] also present work on middleware for distributed context-aware systems. The work is based on a layered architecture for context-aware systems that consists of a context sensor and actuator layer (layer 0), a context processing components layer (layer 1), a context repositories layer (layer 2), a decision support tools layer (layer 3), and finally an application components layer (layer 4). The decision support tools layer is also called adaptation layer and contains user preferences to be applied for adaptation. In this middleware, the aim is not only to decouple the context management from applications, but also to at least partially decouple user preference management from applications. This is done in assisting context-aware applications with making context-based decisions on behalf of the user. For this reason, the decision support tool layer (layer 3) provides preference repositories where applications can store related preferences. In doing so, the system also aims to make preferences sharable and exchangeable between applications. Besides a common context model, the work also presents examples of a common preference model, which is used within this middleware layer. However, the presentation of the user preference model and decision support tool is on a very high level without going into detail how the preference model is implemented and how the decision support is carried out. Furthermore, it is not clear how different application-dependent user attribute vocabularies are taken into account and how an overall user profile is structured that includes the addressed user preferences.

2.2.2.3 Discussion

Many of the referenced approaches on user profile management system aim to support service-specific and situation-dependent user preferences, and hence fulfil goal 1. They also partly identify the need to include an application-independent selection functionality that evaluates the best matching user preferences concerning a certain user situation, e.g. [61] and [62]. However, they do not show mechanisms or algorithms that are required for the actual implementation of such selection functionality. Solely [64] describes a rule-based approach that is restricted to notification preferences. Hence, goal 4 is fulfilled for some works, but for many works it is difficult to find details on the actual functioning of this module.

Furthermore, many approaches do not consider the diversity of user attribute vocabularies. This means that they usually enforce a central user attribute vocabulary without providing a means for application-specific extensibility, sharing or interoperability of user data, which we desire in this thesis with goal 2. In contrast to this, the requirements already mentioned for this thesis are a concrete mechanism for the selection of matching situation-dependent user preferences as well as a means to manage application-specific user data that adheres to arbitrary individual user attribute vocabularies.

Existing research for context-aware systems in the area of ubiquitous computing usually focuses on context gathering, processing, reasoning and provisioning steps. On the one hand a common context model is used to decouple applications from context gathering and subsequent steps. In doing so, the application-independent context-management infrastructure serves arbitrary applications with meaningful context information. However, the user preference model is usually application-specific. This means, applications usually request required context information from the context-management infrastructure, afterwards process this information together with user profile information, and finally customises its services. In this case, the complexity of the selection of matching user preferences has to be addressed by the application developer. Furthermore, this approach does not support sharing of user data between different applications. Solely Henricksen et al. addresses this issue by adding a decision support tool layer to the infrastructure that assists applications with context-based decisions, i.e. selection functionality. However, concrete mechanisms are also not addressed in this approach.

2.2.3 Ontologies for User Situations

This sub-section includes related work on ontologies for user situations such as user location and user activity. Ontologies for user situations are needed in order to describe user situations, receive these descriptions as input to the user profile selection mechanism and to process them. This sub-section is also closed with a discussion.

The ontologies for expressing user context or user situations are analysed based on their capability to support the following goals as stated in section 1.3 and repeated here:

Goal 1:

- Design and implementation of a user profile management system that is capable of managing the following kinds of user data
    - d. Situation-dependent user data such as notification preferences for incoming new messages related to different situations

Here, goal 1 means that the ontology for user context or user situation should be designed in a way that a situational condition in the user profile can be expressed by using definitions from this user context or user situation ontology. Hence, these user context and situation ontologies need expressive high-level statements about the user's context. E.g. it is not sufficient here to have low-level expressions of GPS locations such as *15.23423 degrees* but expressions such as *Room-12*.

Goal 3:

- Design and implementation of a user profile management system that supports automatic service personalisation
    - a. Design and implementation of a user profile selection module as sub-module of the user profile management system that enables automatic selection of matching situation-dependent user preferences
    - b. Design and implementation of a module to receive and process context parameters as input to the user profile selection module

Here, goal 3 means that user context or user situation ontologies and the corresponding context instances need to be made available to the user profile management system and usable for reasoning purposes to select the matching situation-dependent user preferences.

2.2.3.1 Ontologies

In the course of the design of the Context Broker Architecture (CoBrA) [76], an ontology for context-aware pervasive computing environments was created. This ontology contains among others place related classes such as Campus, Building, Room and Stairway, agent related classes such as Agent, Person and Role, agent related location context such as ThingInBuilding and PersonInBuilding, and agent related activity context such as PresentationSchedule, Event, and EventHappeningNow. Furthermore, this ontology also specifies properties such as latitude and longitude for place related classes. This means, that goal 1 and goal 3 is in principal met. However, this ontology only contains few high-level user situation concepts, which are not sufficient but could be extended for our purposes.

The Upper Level Context Ontology (ULCO) [77] and the Standard Ontology for Ubiquitous and Pervasive Applications (SOUPA) [78] also aim to provide context models for pervasive computing environment. ULCO provides a set of basic space concepts common across different environments. The focus is on three classes of real-world objects (user, location, and computing entity) and one class of conceptual objects (activity) that characterise smart spaces. These objects form a skeleton of a contextual environment when linked together. Application developers can use these high-level ULCO concepts and extend it with detailed concepts and properties based on application requirements. SOUPA is designed in a similar way. It consists of two sets of ontology documents, the SOUPA Core and the SOUPA Extension. The core consists of vocabularies for expressing concepts that are associated with person, agent, belief-desire-intention, action, policy, time, space, and event. The extension is defined with two purposes. Firstly, an extended set of vocabularies for supporting specific types of pervasive application domains can be defined. Secondly, the extension aims to demonstrate how to define new ontologies by extending the core. Hence, this means, that goal 1 and goal 3 is in principal met. However, ULCO as well as SOUPA only provide high-level user situation concepts. The definition of detailed concepts is the task of application developers. In contrast to this, the context ontology needed for this thesis should provide detailed yet application-independent concepts for a common understanding throughout the service platform.

In [79], situational reasoning on ontological descriptions in context-aware applications is studied. The work does not show the detailed modelling of the underlying ontology, but give insight into logical foundations of the ontology language OWL [20] and examines how this modelling language can be used to express user situations. It also addresses tool support for OWL-based reasoning and illustrates ontology and reasoning support with a usage scenario.

2.2.3.2 Discussion

Many ontologies for pervasive computing environments such as [76] [77] [78] provide kind of a core ontology with high-level concepts for different types of information that are relevant for these environments. Modelling of detailed concepts, on the other hand is understood as application-specific and should be done by application developers. This means, that goal 1 and goal 3 is in principal met. In service platforms, however, context processing is carried out by a common context management framework that serves arbitrary applications. In order to support these applications with precise context-awareness features, also detailed information such as user situations have to be expressible, i.e. an ontology with detailed modelling concepts is needed. The ontology should for example include space concepts down to single rooms such as an office or a meeting room. Work in [80] and [81] already showed that deriving indoor user locations down to a single room is feasible.

Furthermore, considering space ontologies, most of them focus on representing space concepts from a pure spatial point of view. They e.g. define a room and a building concept. Contrary to this, we think that for our purposes, space concepts should also be modelled according to the function of individual spaces. That is, rooms should be modelled as meeting rooms, offices, printer spaces, presentation spaces, etc., and buildings should be modelled as office buildings, factory buildings and private homes. In addition, space concepts should be distinguished between whether they are in a private, business or public environment. This is because the user’s behaviour differs on whether she is in a private, business or public space, and whether a room is an office, a meeting room, a laboratory or a lounge. Sophisticated situation-dependent personalisation of applications can only be achieved in supporting such features.In this regard , we could not find a matching context ontology.

2.3 Summary

In section 2.1, we have first depicted the fundamentals for this thesis. We have provided definitions for important terms such as the terms user context and user profile, we have depicted basics about service platforms for networks beyond 3G and user profile management systems for service platforms. As our focus is on the selection of matching situation-dependent user data, i.e. a sub-functionality of the user profile management system, we have also provided an overview of Semantic Web technologies that provide the base for the developed selection mechanism.

Section 2.2 contains related work on user profile schemas and ontologies, user profile management systems, context-aware systems and ontologies for user situations. Work published has been discussed and evaluated concerning the capability of fulfilling the requirements for user profile management in service platforms. The corresponding sub-sections have been closed with an overall discussion and have shown that existing approaches are not sufficient in order to reach our intended goals.

In the forthcoming chapters, we first describe our approach for a user profile management system and the related user profile structure. Subsequently, we introduce our ontology specification for the used user profile structure. We then also present our ontology for user situations that is aligned with the finally introduced user profile selection mechanism, which represents the main objective of this thesis.

3 Framework for User Profile Management

In this chapter, we first depict our concept for a user profile structure. This structure enables the definition of user sub-profiles that can be application-specific and optionally also situation-specific. We also evaluate this user profile structure concerning runtime performance for search tasks.

In the second part of this chapter we describe the user profile management framework that handles the user profiles. This framework is depicted by means of its functional sub-modules and their interaction. In addition, the communication with the context management framework of the same service platform is also described that provides information on the user's situation.

3.1 User Profiles

In the motivation section 1.1, we have already described what kinds of user data have to be considered for user profile management in service platforms. In particular, there are user data required by platform services, e.g. for accounting and billing, and also user data required by various end-user services such as innovative messaging services and location-based services. All this user data can be divided into general user data such as user name, date of birth, address, credit card details, etc. which may be required by all or many services, and into service-specific user data and preferences such as ring tone preferences for telephony, which are only required by a certain service or a certain group of services. Furthermore, user data can also be divided into situation-dependent user data. For example, a user could have different preferences for the notification of incoming news messages. While the user is at home, she may want to be notified of incoming messages immediately, whereas in a business meeting, she may want the notification to be postponed until after the meeting.

We also presented existing solutions to user profile standards and specifications in section 2.2.1.1, and we discussed these existing approaches in section 2.2.1.3 subsequently concerning the ability to match the desired requirements for service platforms. As we concluded that none of the existing approaches fully covers the desired requirements, we depict in the following our approach on a user profile structure for user profiles in service platforms. In addition, we also provide an analytic evaluation for search tasks concerning the depicted user profile structure.

3.1.1 User Profile Structure

The conceptual approach for our user profile structure is shown in Figure 9. As already mentioned in the definition of the term user profile in section 2.1.1, a user profile is the collection of all user data about a particular user apart from context. Thus, there is exactly one user profile for each user. This overall user profile contains application-specific parts. However, the term application-specific does not necessarily mean that such a part is only specific for exactly one application. It could also be specific to a group of applications. In particular, such a group of applications would have to be identifiable by a certain common application identifier and it would have to adhere to the same user attribute semantics to be able to process the related user data. Hence, in case all developers of applications for a service platform agreed on a common semantic for user attributes, only one application-specific part could be sufficient, not considering any possibly required permission rights to query and manipulate user data, which still may be different for different applications. As such an agreement on a common semantic between service developers is usually not the case, different application-specific parts are supported. This makes the use of the user profile structure flexible, since this is an optional but no mandatory feature.

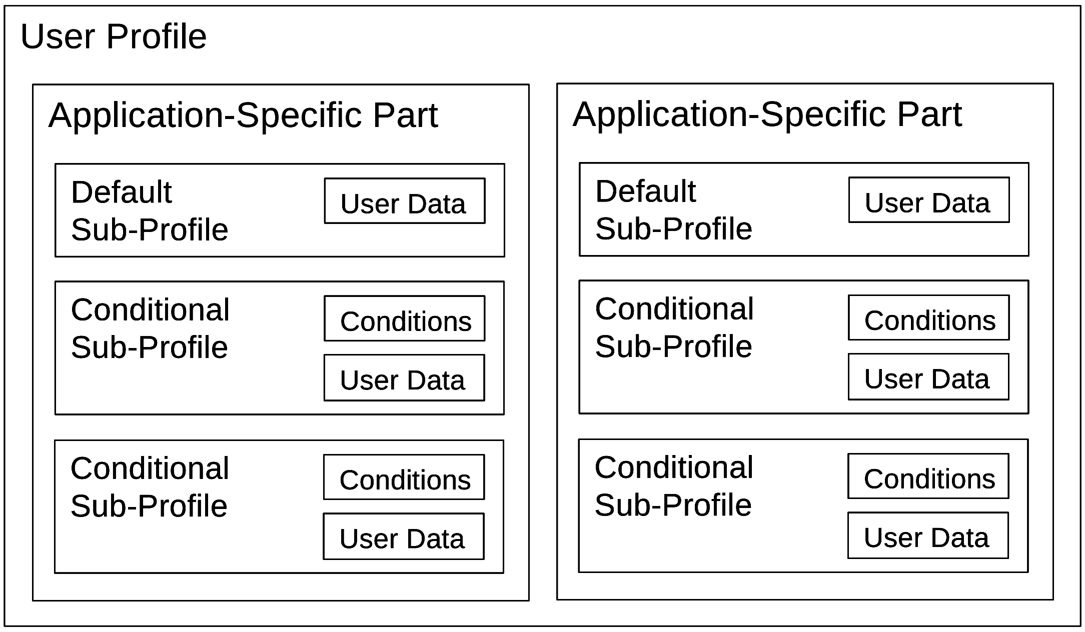


Figure 9: User Profile Structure

An application-specific part contains a mandatory default sub-profile with default user data as well as one or more optional conditional sub-profiles with conditional user data. Default user data is always valid except in those user situations, for which there is a conditional sub-profile. In this case, default user data is superseded by conditional user data. The default sub-profile, in particular the default user data of an application-specific part could be compared to the domain submodel in the Doppelgänger User Modeling System [43]. The conditional sub-profile, in particular the conditional user data could be compared to a conditional submodel. The Doppelgänger User Modeling System has already been described in section 2.2.1.2.

Our approach of defining default sub-profiles and conditional sub-profiles within an application-specific part is for sure not the only possible approach for enabling situation-dependent user data to be added to user profiles. A counter proposal could be to define only one common sub-profile in an application-specific part. In this sub-profile, each single user attribute would have to be specified as being conditional or not. However, there are two reasons, why we prefer the distinction between default sub-profiles and conditional sub-profiles. Firstly, this distinction adds some structure and order to the application-specific part and related user data respectively. This is important for the visualisation of user profile content to the user via a user editor, and in particular should the application-specific part include a huge list of user data. Secondly, runtime performance for search tasks can be improved in case user data are clustered in default and conditional sub-profiles, as described in section [3.1.2].

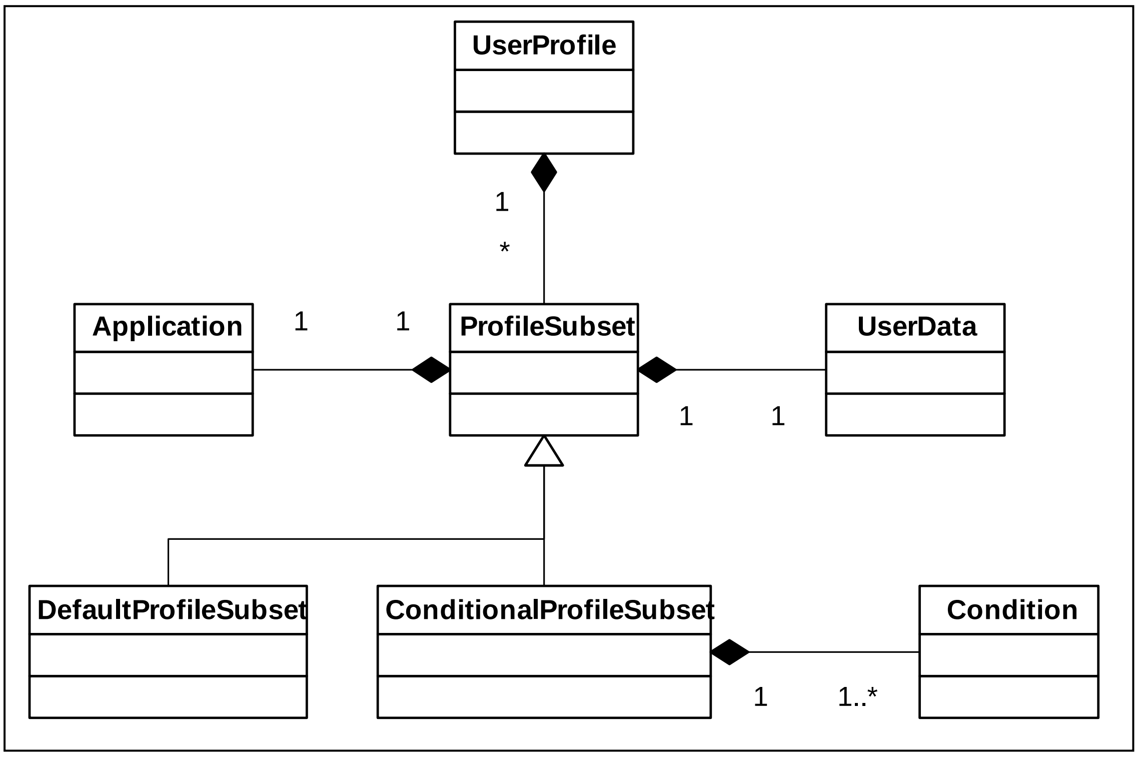


Figure 10: User Profile Class Diagram

The implementation of the described user profile structure is shown in Figure 10 by means of a programming language independent class diagram. The classes and relations between classes are explained as follows:

User Profile Class: This class represents the user profile of a certain user. It is associated with arbitrary many Profile Subsets. The type of association between a User Profile and a Profile Subset is a composition, as a Profile Subset belongs to exactly one User Profile and as the Profile Subset does not exist without the related User Profile.

Profile Subset Class: The term user profile subset has already been introduced as a synonym for user sub-profile in section 2.1.1. This class is an abstract class that has the two specialisations Default Profile Subset and Conditional Profile Subset. An own class for application-specific parts is not implemented. Instead, the Profile Subset class has a 1 to 1 composition association with the Application class. Consequently, the Profile Subset must always be associated with an application. Furthermore, the Profile Subset must also be associated with a User Data object.

Application Class: This class primarily represents an application identifier. This identifier could also be an identifier for a group of applications.

User Data Class: This class represents an object that includes the actual user data. It is described in more detail below.

Default Profile Subset Class: This class represents the above introduced default sub-profile of an application-specific part.

Conditional Profile Subset Class: This class represents the above introduced conditional sub-profile of an application-specific part. It is associated with at least one condition. In case of several conditions, our implementation interprets these conditions as conjunct to each other.

Condition Class: This class represents a situational condition. Such a condition could express a user location, user activity, time of day or other types of user situations. It is described in more detail below.

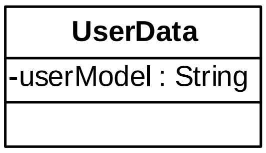


Figure 11: User Data Class Diagram

Figure 11 shows the User Data class in more detail. It basically comprises one attribute of type String. This attribute contains the whole user data, i.e. an instance of a user model, for the related profile subset. However, the content of the String object has to follow RDF [17] syntax. In particular, this attribute contains an RDF/XML serialisation [82] of an RDF graph. There are several formats for RDF serialisations, whereof the RDF/XML serialisation is one of most often used ones, as it follows XML [13] syntax. This approach of encoding user data is very flexible. There is no pre-defined list of user attributes, which can be added to a profile subset. Instead, the RDF/XML serialisation enables the inclusion of arbitrary application-specific user attributes. The underlying ontology, which defines the meaning of the RDF/XML encoded user model instance, is not considered here. However, the related ontology has to be accessible by the user profile management framework should transformations of user model instances adhering to different user attribute ontologies be enabled. This is explained in more detail in section 3.2.

```
<rdf:RDF
        xmlns:upr="http://www.example.org/profile.owl#"
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
        xmlns:owl="http://www.w3.org/2002/07/owl#"
        xmlns:daml="http://www.daml.org/2001/03/daml+oil#">
  <upr:UserModel rdf:about="http://www.example.org/profile.owl#SandyUserModel">
          <upr:firstname>Sandy</upr:firstname>
          <upr:lastname>Smith</upr:lastname>
          <upr:age>25</upr:age>
          <upr:email>Sandy.Smith@example.org</upr:email>
  </upr:UserModel>
</rdf:RDF>
```

Figure 12: Example for an RDF/XML Serialised User Model Instance

Figure 12 shows an example for an RDF/XML serialisation of a small user model instance. The identifier of the user model instance is *SandyUserModel*. It includes the user attributes *firstname*, *lastname*, *age* and *e-mail address*. In addition, the root element *rdf:RDF* contains several XML namespace (XMLNS) definitions.

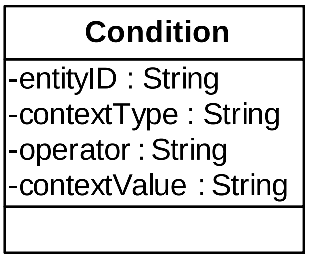


Figure 13: Condition Class Diagram

Figure 13 depicts the Condition class consisting of four attributes. The *entityID* identifies an entity such as a user, a group of users or another object the situational condition is related to. The *contextType* identifies the type of context such as user location, user activity, time of day, day of week, etc. The *contextValue* represents the value of a context type, such as a particular room should the context type be location, a certain activity, a certain day of week, etc. The operator relates the *contextValue* with the *contextType*. In case the context type is location, the operator could e.g. be *is-connected-to*, *is-adjacent-to*, *is-part-of*, *is-nearby*, *equals*, etc. Should the context type be temperature, the operator could e.g. be *is-higher-than*, *is-lower-than*, *equals*, etc. A similar representation is used in the Gaia meta-operating system [65], which has already been introduced in section 2.2.2.2. In Gaia, the structure (<ContextType>, <Subject>, <Relater>, <Object>) is used for representing context in general and is similar to a simple clause in the English language of the form <subject> <verb> <object>.

The concrete entity identifiers, context types, operators and context values being available depend on the system, i.e. the service platform, in which the user profiles are applied. In particular, they depend on the modelling of the context representation, i.e. the context ontologies in the related system. Such an example implementation of context representation and related reasoning is shown in chapter 4 and 5. However, the combination of these four attributes enables very expressive situational conditions.

Table 1: Examples for Situational Conditions

EntityID	***ContextType***	***Operator***	***ContextValue***
Bob	location	is-connected-to	Room-220
Room-15	temperature	is-higher-than	30
Alice	activity	equals	Watching-TV
Chris	location	is-within	Premises-2

Some few examples are shown in Table 1. The depicted structure of situational conditions is not only very expressive, but also easily understandable for non-technical users. In combination with a profile editor, these four attributes that form a situational condition could be shown to the user as select lists. That is, for each of the four attributes, the user does not have to fill out a blank text field, but can select from a pre-defined list. For this purpose, the meaning and the values for all four attributes have to be easily understandable. The biggest challenge may be to provide easily understandable operators. However, the examples in Table 1 show that this is feasible.

3.1.2 Runtime Performance for Search Tasks

In this section, we show the advantage of clustering user data into profile subsets concerning runtime performance for different search tasks. In particular, there are basically two different search tasks:

Search Task 1 (Profile Subset Search): An application requests the complete set of matching user data, i.e. a profile subset. This means that it requests all user data that matches the current user's situation. Alternatively, if the application has subscribed to the user profile management system beforehand, the application is notified by the user profile management system of the complete set of matching user data.

Search Task 2 (Single User Record Search): An application requests a single piece of matching user data, i.e. a user record. This means that it requests a single piece of user data that matches the current user's situation. Alternatively, if the application has subscribed to the user profile management system beforehand, the application is notified by the user profile management system of the matching piece of user data.

The forthcoming analytic evaluation assumes that each user profile contains only one application-specific part. In the evaluation we assume that all applications adhere to the same semantics and hence can understand and share required user data by means of a common application-specific part. However, the depicted evaluation can afterwards easily be adapted to the case in which different application-specific parts are needed. In order to carry out the analytic evaluation, we need some additional definitions beside the definitions in section 2.1.1.

User Record: A user record is a piece of information about a particular user. A user record can either be a default user record or a conditional user record.

Default User Record: A default user record consists of a user attribute and the related value.

Non-Conditional User Record: Synonym for default user record.

Conditional User Record: A conditional user record consists of a situational condition, a user attribute and the related value.

Situation-Dependent User Record: Synonym for conditional user record.

Finally, we also need several assumptions in order to carry out the analytic evaluation.

Assumption 1: The user attribute of a default user record and the combination of situational condition and user attribute of a conditional user record is unique within a profile subset. That is, each user attribute and combination of situational condition and user attribute respectively only occurs once within a profile subset.

Assumption 2: If the application-specific part of a user profile is separated into different profile subsets, then there is always exactly one default profile subset.

Assumption 3: If the application-specific part of a user profile is separated into different profile subsets for different situational conditions, then there is always only one profile subset for each distinct situational condition.

Assumption 4: If the application-specific part of a user profile is separated into different profile subsets, then each profile subset (default and conditional ones) contains the same number of user records.

As the forthcoming analytic evaluation contains a comparison of three different user profile structures, these user profile structures are introduced now.

Condition	***Attribute***	***Value***
-	First Name	Bob
-	Date of Birth	12 Jan 1970
Home	Ring Tone	On
Meeting	Call Forward	On
-	Vibration Alert	On
Meeting	Ring Tone	Off
-	Call Forward	Off
Home	SMS Notification	Off
-	Ring Tone	On
Home	Call Forward	Off
Meeting	SMS Notification	On
...	...	...

Figure 14: One Overall Profile Subset in Approach 1

Approach 1: In this user profile structure approach, an application-specific part of a user profile consists of one overall homogeneous profile subset as shown in Figure 14. This profile subset includes the whole set of user records, i.e. of condition-attribute-value triples. As not all user records are conditional, i.e. as some user records represent default user records, the condition could simply be empty. In this approach, the result of the profile subset search (search task 1) provides a virtual profile subset, i.e. the subset of all user records with a certain condition.

Default profile subset

Attribute	***Value***
First Name	Bob
Date of Birth	12 Jan 1970
Ring Tone	On
Call Forward	Off
...	...

Conditional profile subset

Condition	***Attribute***	***Value***
Home	Ring Tone	On
Home	Call Forward	Off
Office	SMS Notification	On
Meeting	Ring Tone	Off
Home	SMS Notification	Off
Meeting	Call Forward	On
...	...	...

Figure 15: Two Profile Subsets in Approach 2

Approach 2: In the second user profile structuring approach, the application-specific part of a user profile consists of two separate concrete profile subsets, see Figure 15. In particular, it consists of a default profile subset and a conditional profile subset. The default profile subset only contains default user records, and the conditional profile subset contains all conditional user records. In this approach, the result of the profile subset search (search task 1) either provides the concrete default profile subset as depicted in Figure 15 or a virtual conditional profile subset consisting of all conditional user records with a certain condition. This virtual conditional profile subset is a subset of the concrete existing conditional profile subset shown in Figure 15.

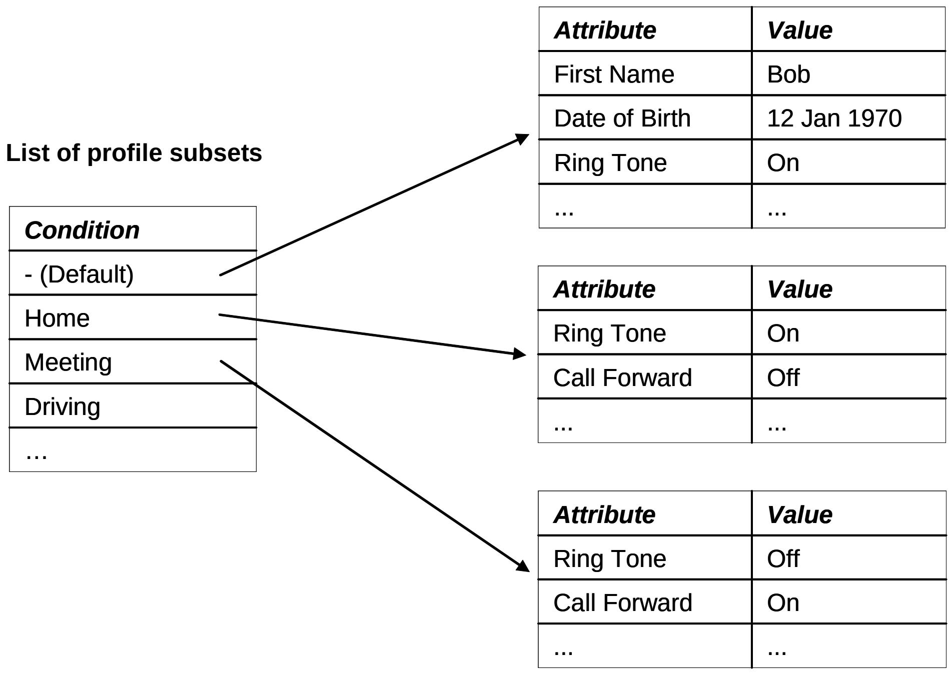


Figure 16: One Profile Subset for Each Distinct Condition in Approach 3

Approach 3: In this approach, the application-specific part of a user profile is separated into a default profile subset and in addition into one conditional profile subset for each distinct situational condition, see Figure 16. In this approach, the result of the profile subset search (search task 1) either provides the concrete default profile subset as depicted in Figure 16 or a concrete conditional profile subset as shown in Figure 16. Thus, in this approach no virtual profile subset has to be searched. This approach represents our approach that we have introduced in section 3.1.1.

The analytic comparison of these three user profile structuring approaches is based on two well-known example data structures, Sorted Linked List and AVL Tree [83] [84]. Linked Lists are one of the fundamental data structures in computer science. They consist of a sequence of nodes, each containing one link pointing to the next node. For the Sorted Linked List, we assume that user records are not primarily sorted by conditions. AVL Trees, on the other hand, are self-balancing binary search trees. In AVL Trees, the heights of the two child sub-trees of any node differ by at most one. Therefore, an AVL Tree is also height-balanced. For the AVL Tree, we also assume that the nodes are not primarily sorted by conditions, as probably is the case in usual user profile management systems.

The results of the analytic comparison are expressed by means of the number of needed comparison steps for finding the matching user records, which is distinct from the number of comparison steps for fetching the matching user records. For this purpose, the Big O notation is used, also known as the Landau notation [83] [84]. The Greek letter O (Omicron) is used to describe the asymptotic upper bound for the magnitude of a function, Ω (Omega) is used to describe the asymptotic lower bound and Θ (Theta) is used to describe the case in which the magnitude of a function is in O and Ω at the same time.

The results for all three user profile structuring approaches, using both list structure-based as well as tree structure-based data organisation, are depicted for both above introduced search tasks. Furthermore, results are depicted for average-case and worst-case runtime. The results for search task 1 (Profile Subset Search) are shown in Table 2, whereas the results for search task 2 (Single User Record Search) are shown in Table 3. Both results are explained in more detail below. In the following, the variables used in the tables are explained:

m: overall number of user records in an application-specific part of a user profile (m = p+q)
p: number of default user records (p = m/(r+1) <= m; see assumption 4)
q: number of conditional user records (q = p*r = m*r/(r+1) <= m; see assumption 4)
r: number of different conditions (r <= q <= m)

Table 2: Runtime Comparison for Profile Subset Search (Search Task 1)

Profile Subset Search		***Approach 1***		***Approach 2***		***Approach 3***	
		List	***Tree***	***List***	***Tree***	***List***	***Tree***
Default Profile Subset	***Avg***	$\Theta(m)$	$\Theta(m)$	$\Theta(1)$	$\Theta(1)$	$\Theta(r)$*	$O(\log_2(r))$ *
	Worst	$\Theta(m)$	$\Theta(m)$	$\Theta(1)$	$\Theta(1)$	$\Theta(r)$*	$O(\log_2(r))$ *
Conditional Profile Subset	***Avg***	$\Theta(m)$	$\Theta(m)$	$\Theta(q)$	$\Theta(q)$	$\Theta(r)$	$O(\log_2(r))$
	Worst	$\Theta(m)$	$\Theta(m)$	$\Theta(q)$	$\Theta(q)$	$\Theta(r)$	$O(\log_2(r))$

Comparison for Profile Subset Search (Search Task 1): The results of the User Profile Subset Search are shown in Table 2. In approach 1, independent of the used data organisation, all user records have to be analysed whether they match the queried condition or not, hence Θ(m) in all cases. This is because the user records are not sorted by their conditions and because we do not know where the related user records can be found in the list and the tree respectively.

In approach 2, searching for the default profile subset requires exactly one comparison step, as we know after the first comparison step that we either found the concrete default profile subset or the concrete conditional profile subset, hence Θ(1). Querying for the virtual conditional profile subset requires one comparison step for finding the concrete conditional profile subset, and afterwards Θ(q) comparison steps independent of the used data organisation, because all conditional user records in the concrete conditional profile subset have to be analysed. This is because the conditional user records are not sorted by their conditions and because we do not know where the related conditional user records can be found in the list and the tree respectively.

Approach 3 is slightly more complex. The average-case in the list-based data organisation is in Θ(r). This is because the average case needs (1+2+…+r+(r+1))/(r+1) = ((r+1)*(r+2)/2)/(r+1) = (r+2)/2 comparison steps. The results for the tree-based search require more complex computations, which can be found in [83] and [84]. The asterisk denotes that the actual implementation could be done in Θ(1), because we could introduce a distinct data type for the unique default profile subset as shown in Figure 10. Hence, one comparison step would be enough should an appropriate organisation of profile subsets be considered.

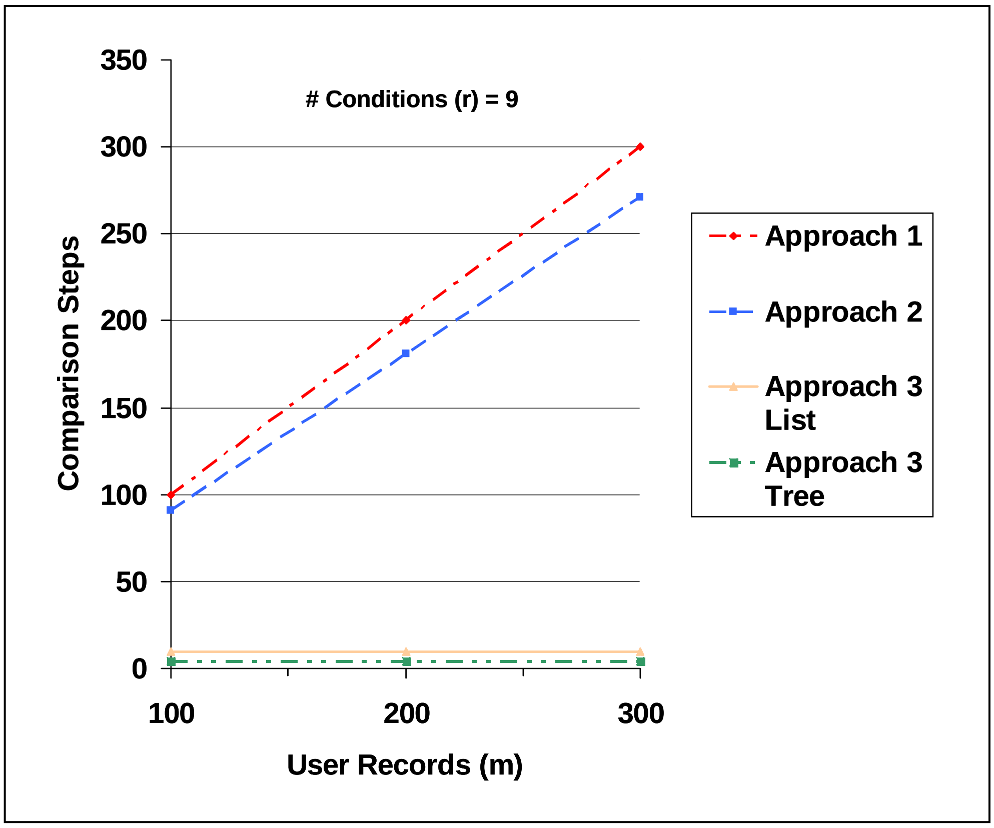


Figure 17: Worst Case Search of Conditional Profile Subset with 9 Different Conditions

Figure 17 shows the quantitative results for the worst case analysis of searching for a conditional profile subset in case nine different conditions exist. Considering the default profile subset, there are ten profile subsets altogether. Approach 1 and approach 2 both are valid for list-based as well as tree-based data organisation. Both depend on the overall number of user records (m) as variable $q = m*r/(r+1) = m*9/10$. In comparison to this, the needed comparison steps in approach 3 are constant for both, list-based and tree-based approach, as they depend on the number of different conditions (r). As can be seen in Figure 17, approach 3 outperforms the other approaches concerning the needed comparison steps for finding a conditional profile subset.

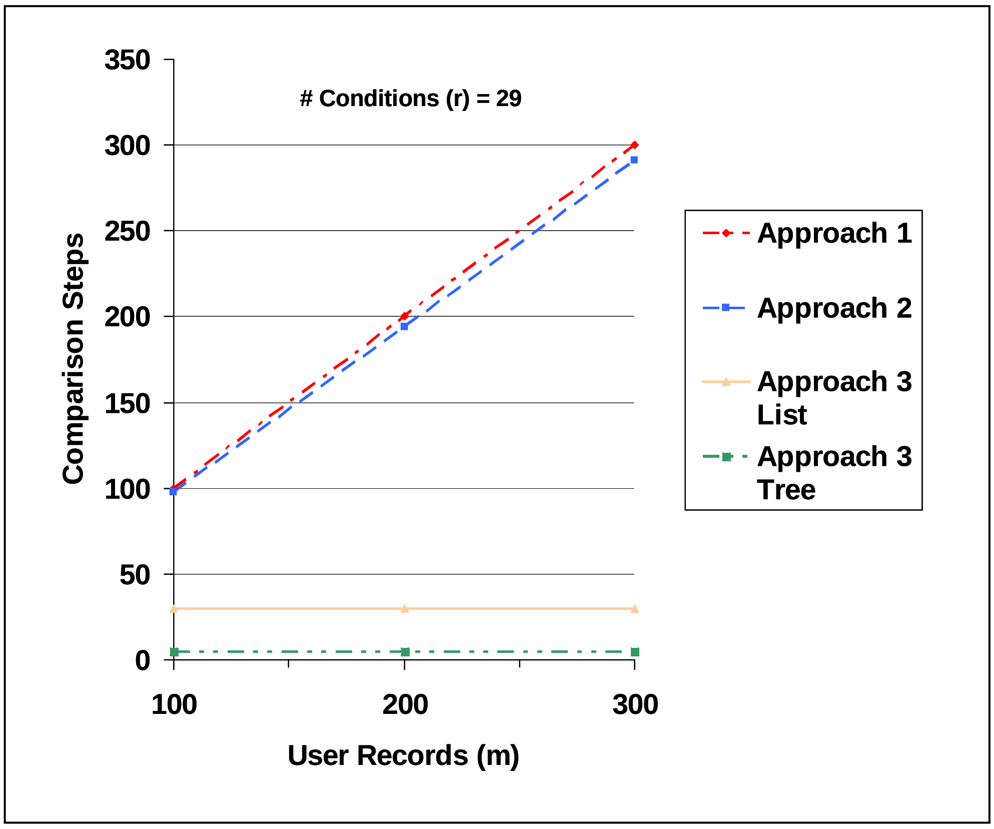


Figure 18: Worst Case Search of Conditional Profile Subset with 29 Different Conditions

Almost exactly the same result can be seen in Figure 18, which shows the results for 29 different conditions, i.e. 30 profile subsets altogether. The main difference is that the tree-based data organisation clearly outperforms the list-based data organisation in approach 3. The average case analysis also provides the same overall results. There are no differences for approach 1 and 2 in the average case analysis. In comparison to this, approach 3 even requires less comparison steps in average. The functions applied in the worst-case graphs in Figure 17 for m >= 10 and in Figure 18 for m >= 30 are as follows:

Approach 1: $y = m$
Approach 2: $y = 1+q = 1+m*r/(r+1)$
Approach 3 List: $y = r+1$
Approach 3 Tree: $y = \log_2(r+1)$

Table 3: Runtime Comparison for Single User Record Search (Search Task 2)

Single User Record Search		***Approach 1***		***Approach 2***		***Approach 3***	
		List	***Tree***	***List***	***Tree***	***List***	***Tree***
Default User Record	***Avg***	$\Theta(m)$	$O(\log_2(m))$	$\Theta(p)$	$O(\log_2(p))$	$\Theta(r+p)$*	$O(\log_2(r)+\log_2(p))$ **
	Worst	$\Theta(m)$	$O(\log_2(m))$	$\Theta(p)$	$O(\log_2(p))$	$\Theta(r+p)$*	$O(\log_2(r)+\log_2(p))$ **
Conditional User Record	***Avg***	$\Theta(m)$	$O(\log_2(m))$	$\Theta(q)$	$O(\log_2(q))$	$\Theta(r+q/r)$	$O(\log_2(r)+\log_2(q/r))$
	Worst	$\Theta(m)$	$O(\log_2(m))$	$\Theta(q)$	$O(\log_2(q))$	$\Theta(r+q/r)$	$O(\log_2(r)+\log_2(q/r))$

Comparison for Single User Record Search (Search Task 2): The results of the Single User Record Search are shown in Table 3. In approach 1, the average and worst-cases for the list-based data organisation in both default and conditional case are in Θ(m). The average-cases are already in Θ(m), because these cases need (1+2+…+(m-1)+m)/m = (m*(m+1)/2)/m = (m+1)/2 comparison steps.

The same computation also holds for the list-based data organisation in approach 2, only applying variable p and q respectively. In these cases one additional comparison step is needed in order to first find the default profile subset and conditional profile subset respectively. The average and worst-cases for the tree-based data organisation for both approach 1 and 2 can be read in [83] and [84]. We assume in this case that we can take advantage of the sorting of tree nodes. As assumed above, tree nodes are not primarily sorted by conditions, but they may be primarily sorted by the attribute names of the user records. In this search task, contrary to search task 1, we know the attribute name of the user record as we search for a very specific user record.

The evaluation for approach 3 is again slightly more complex. For the list-based data organisation, the number of comparison steps in average-case and worst-case is in Θ(r+p) and Θ(r+q/r) respectively, where q/r is the number of conditional user records per condition based on the above described assumption 4. The computations result from first finding the matching profile subsets out of r+1 profile subsets, and afterwards finding the matching user record out of p default user records for the default profile subset and q/r conditional user records for a conditional profile subset respectively. Furthermore, the average-case computation follows the same idea as for approach 1, but with other variables. The single asterisk denotes that the actual implementation can be done in Θ(p) for the same reasons as explained in the results for the profile subset search (search task 1).

The computation for the tree-based data organisation of approach 3 is similar to that for list-based data organisation, only with logarithmic runtime. Details of the computation can again be read in [83] and [84]. The double asterisk denotes that the actual implementation can be done in $O(\log_2(p))$, again for the same reasons as explained in the results for the profile subset search (search task 1).

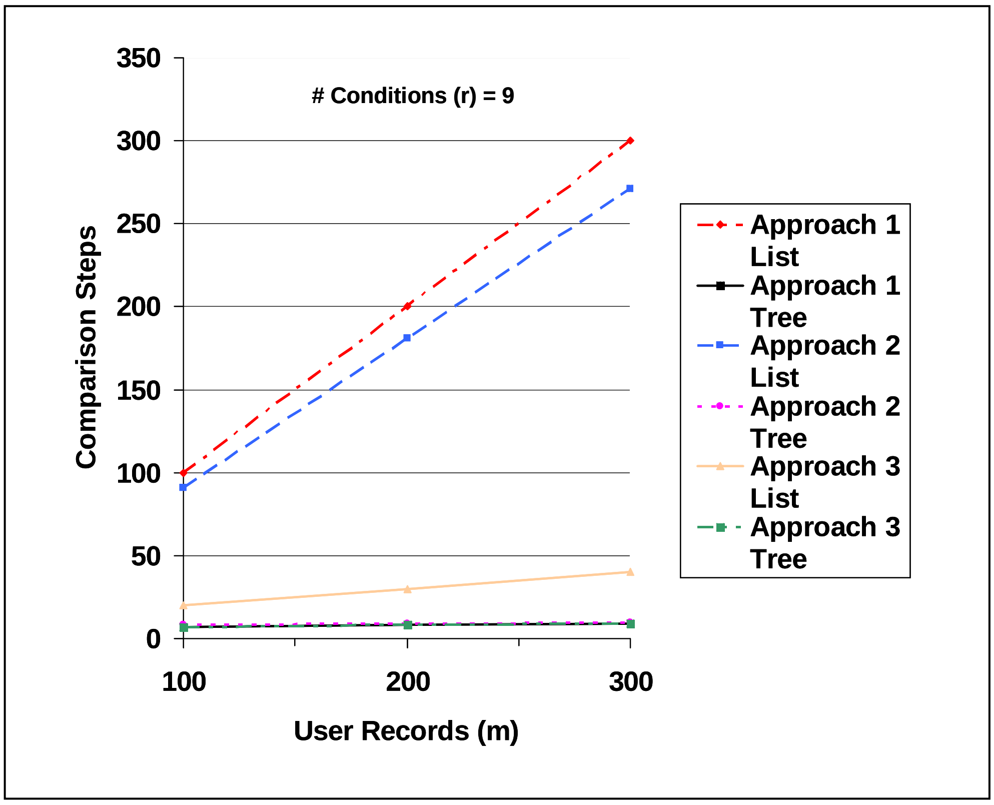


Figure 19: Worst Case Search of Conditional User Record with 9 Different Conditions

Figure 19 shows the quantitative results for the worst case analysis of searching for a conditional user record in case nine different conditions exist. This time, all functions depend on the overall number of user records (m) as variable q depends on variable m. As can be seen in Figure 19, approach 3 outperforms the other approaches concerning the number of needed comparison steps for the list-based data organisation. For the tree-based data organisation on the other hand, the performance of all approaches is almost the same, as $\log_2(r+1)+\log_2(q/r) = \log_2((r+1)*q/r) = \log_2((r+1)*m*r/((r+1)*r)) = \log_2(m)$. The average case analysis would show slightly better quantitative results for the list-based data organisation, but still be in the same magnitude of a function as depicted in Table 3. The functions applied in the worst case graph in Figure 19 for m >= 10 are as follows:

Approach 1 List: $y = m$
Approach 1 Tree: $y = \log_2(m)$
Approach 2 List: $y = 1+q = 1+m*r/(r+1)$
Approach 2 Tree: $y = 1+\log_2(q) = 1+\log_2(m*r/(r+1))$
Approach 3 List: $y = r+1+q/r = r+1+m/(r+1)$
Approach 3 Tree: $y = \log_2(r+1)+\log_2(q/r) = \log_2(r+1)+\log_2(m/(r+1)) = \log_2(m)$

Summary: The results show that the user profile structuring approach 3, which is the approach we have introduced in section 3.1.1, outperforms the other user profile structuring approaches concerning the needed comparison steps especially in the profile subset search (search task 1). It is important to mention that we have only focused on the number of comparison steps. We have not said anything about what is going on within the comparison steps.

Whereas the overall result may not be interesting for single context-aware applications with few user data, it may be important for user profile management systems in service platforms that manage a huge conglomeration of user data. This is especially the case should user profile management tasks be carried out on mobile devices with limited processing power. Consider, for example, that the user profile management system manages user preferences for 10 applications, each application requiring 10 preferences, and the user distinguishing between 10 different situations. Hence, there may be up to 1000 user preferences. Whereas the runtime performance for the required number of comparison steps may still be fast for e.g. straightforward string comparisons, it may get unacceptably slow should ontology processing and reasoning be involved in the comparison steps. As described later in this thesis, our user profile selection mechanism takes advantage of ontology reasoning in the user profile selection process, i.e. the user profile search process. Hence, the introduced user profile structure is an important base for the development of a fast user profile selection mechanism.

3.2 User Profile Management Framework

The user profile management framework is composed of several functional sub-modules, which are easily exchangeable. This framework and the related sub-modules are introduced next. Afterwards, the interaction between the user profile management framework and a context management framework that is also part of the same service platform is explained.

3.2.1 User Profile Management

As shown in Figure 20, the user profile management framework consists of three layers. The main layer is the Management Layer that consists of several modules, the User Profile Management Core, the Profile Subset Management, the User Profile Selection Module, the User Profile Sharing Module and the Context Client Module.

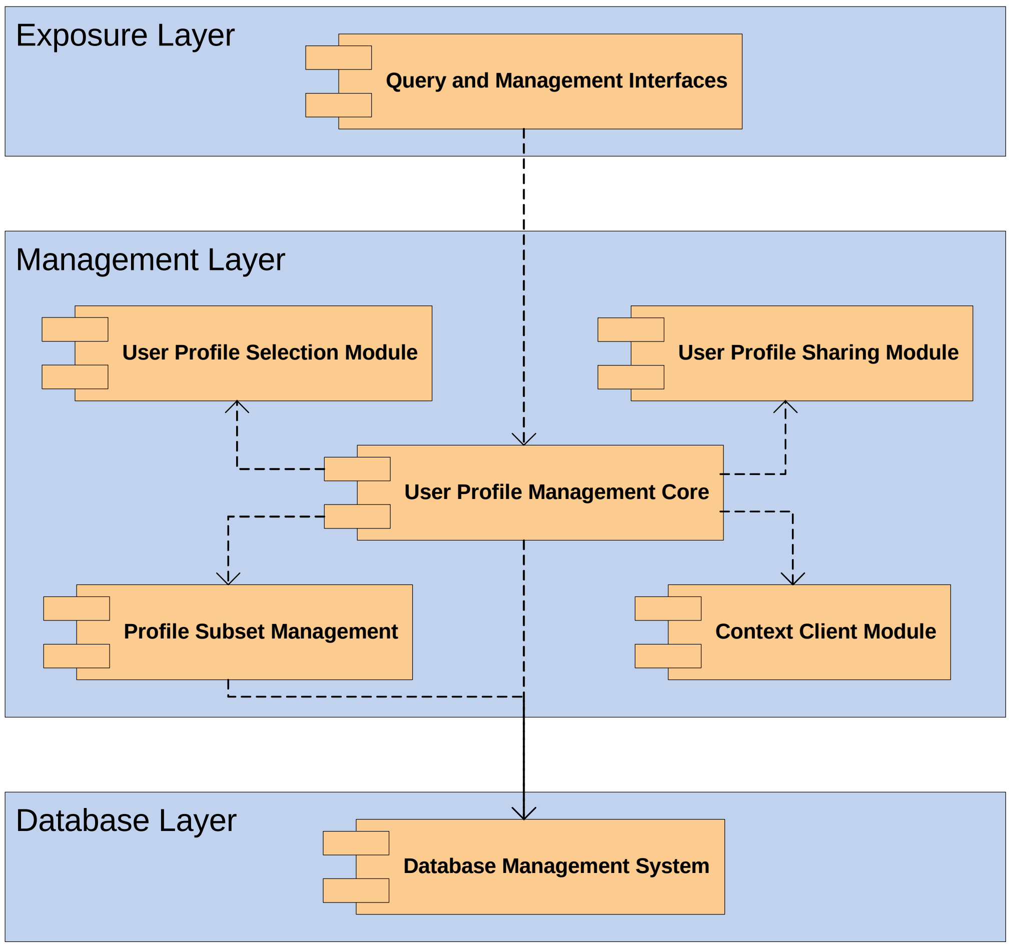

Exposure Layer
Query and Management Interfaces
Management Layer
User Profile Selection Module
User Profile Sharing Module
User Profile Management Core
Profile Subset Management
Context Client Module
Database Layer
Database Management System

Figure 20: User Profile Management Framework

Second, there is the exchangeable Database Layer and finally the Exposure Layer. The Management Layer communicates with the Database Layer for user profile storage and retrieval and the Exposure Layer communicates with the Management Layer for processing client requests. In the following the modules of the Management Layer are described in more detail:

User Profile Management Core: This is the core module of the Management Layer, which organises all processes. It communicates with the Exposure Layer as well as the Database Layer and delegates tasks and client requests to other modules. For example, it delegates the creation of profile subsets to the Profile Subset Management or it delegates user profile queries to the User Profile Selection Module. However, some basic tasks such as profile creation are also covered by this module.

Profile Subset Management: This module is responsible for the whole management related to profile subsets. This includes creation, modification, deletion of profile subsets and profile subset related information, i.e. situational conditions and user data.

User Profile Selection Module: This module processes client queries. It comprises an evaluation functionality that returns the best matching profile subset and user data respectively for the user's current situation. Using this functionality, a client application does not need an own potentially complex evaluation functionality. Instead, the user profile management framework in service platforms covers this functionality for all client applications, at least for those types of context that are processed within the service platform. The user profile selection mechanism that is applied by this module is the central part of this thesis. Details of the developed mechanism are provided in chapter 5.

Context Client Module: This module is also needed for user profile selection tasks. It communicates with other external services, in particular the context management framework of the same service platform, in order to request relevant user context needed for the user profile selection task. For our purposes, we assume that the received user context is provided as a high-level context description, e.g. a certain room or a certain user activity, instead of low-level context descriptions such as GPS (Global Positioning System) information. Furthermore, we assume that the description of the provided user context is provided as an RDF [17] document, that it is based on the context ontology of section 4.3 and that the user context is accurate. That is, we do not deal with probabilities related to user context. The communication with the context management framework is explained in more detail in the next section 3.2.2.

User Profile Sharing Module: This is the module that enables the reuse and sharing of user data between different applications. It provides a means to transform a user model instance that adheres to a particular user attribute schema to a user model instance that adheres to another particular user attribute schema. After a successful transformation, the same user data is available for different applications. Although we identified this functionality as a requirement for user profile management in service platforms, we do not focus on this functionality in this thesis. However, the developed user profile management framework provides an interface for easily adding an implementation of this module. As mentioned in the section on related work 2.2.1.1, the work by Sluijs and Houben [58] [59] could be used to provide this functionality. It should be mentioned that additional data repositories are needed for storing schemas and schema mappings, which is not shown in Figure 20.

The Management Layer is designed in such a way that the User Profile Selection Module, the Context Client Module and the User Profile Sharing Module are easily exchangeable. This makes the user profile management framework flexible as it can easily be adapted to different requirements, e.g. for different service platforms. For example, as the representation of context information could be different in different service platforms, the Context Client Module could be exchanged with one that is adapted to the corresponding context representation. Another example would be the exchange of the User Profile Selection Module in order to apply a different search mechanism. Whereas our search mechanism is based on ontology reasoning, we could also use a search mechanism that is based on rule-based reasoning.

In the following, we provide an activity diagram, i.e. an algorithm, for the user profile query case in order to explain the functioning of the user profile management framework. It should be mentioned that privacy and data security related issues are not covered here. However, in the SPICE (Service Platform for Innovative Communication Environment) project [4] [5] [6], these issues are covered on a service platform level instead of on a component level. In addition, the SPICE related work in [85] introduces a trust framework for service platforms.

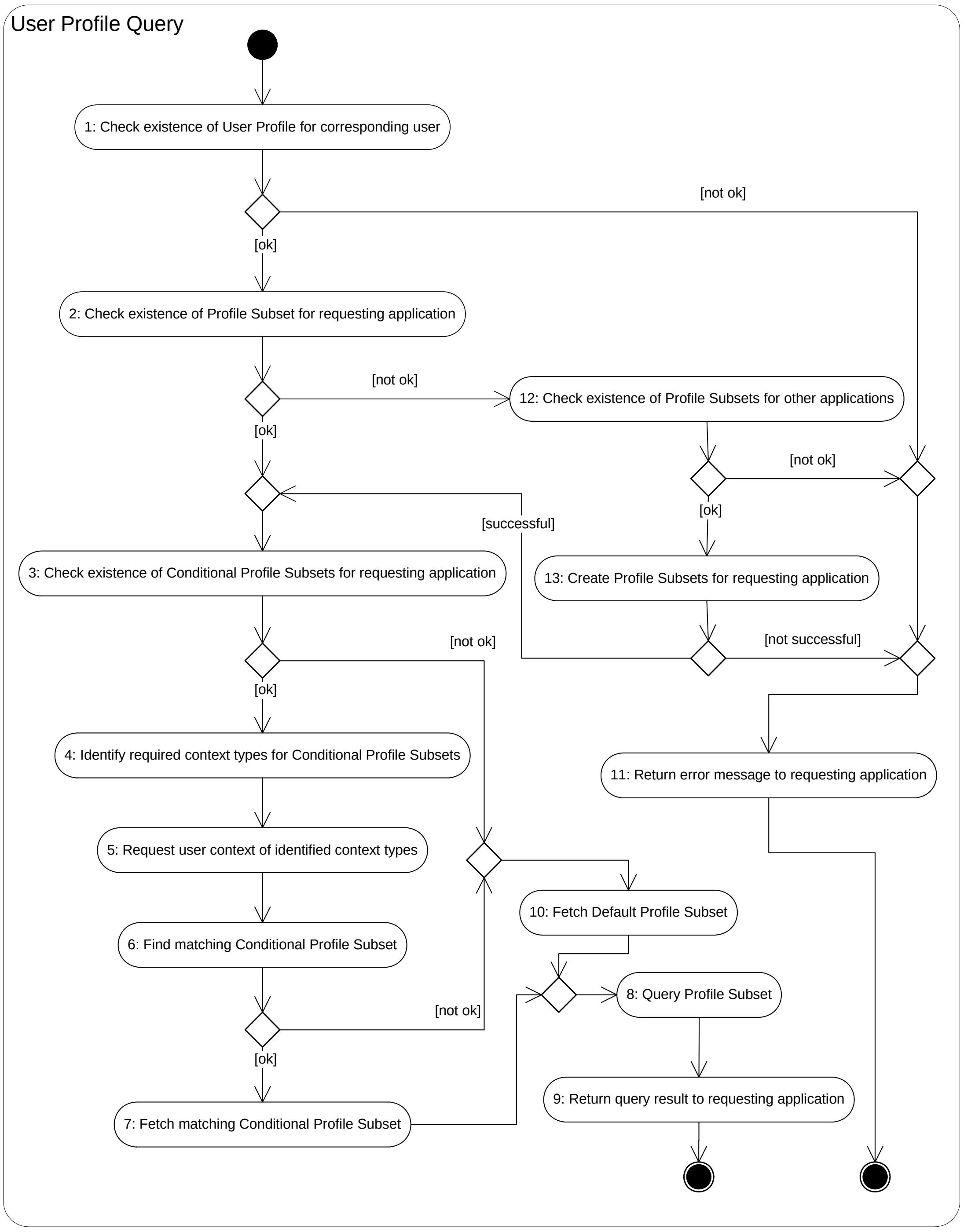

User Profile Query
1: Check existence of User Profile for corresponding user
[not ok]
[ok]
2: Check existence of Profile Subset for requesting application
[not ok]
12: Check existence of Profile Subsets for other applications
[ok]
[not ok]
[ok]
[successful]
3: Check existence of Conditional Profile Subsets for requesting application
13: Create Profile Subsets for requesting application
[not ok]
[not successful]
[ok]
4: Identify required context types for Conditional Profile Subsets
11: Return error message to requesting application
5: Request user context of identified context types
10: Fetch Default Profile Subset
6: Find matching Conditional Profile Subset
8: Query Profile Subset
[not ok]
[ok]
9: Return query result to requesting application
7: Fetch matching Conditional Profile Subset

Figure 21: Activity Diagram for User Profile Query

Figure 21 shows the activity diagram for the user profile query case. In this use case, a client application requests the best matching user data. In particular, the request includes an additional query parameter in order not to get the complete matching profile subset, but specific parts thereof. The activity diagram has one starting point and two end points. One end point represents the success case, i.e. the successful response to the query, and the other end point represents the non-success case, i.e. the query could not be processed e.g. because no user profile existed for the corresponding user. In the following, the activities in Figure 21, which are all numbered, are explained step by step.

Activity 1 (Check existence of User Profile for corresponding user): In this first step, the User Profile Management Core checks whether the corresponding user has a user profile. If this is the case, the user profile query continues. Otherwise, this query ends up in the non-success end point.

Activity 2 (Check existence of Profile Subset for requesting application): In this step, the User Profile Management Core checks whether the user's profile contains at least one profile subset for the requesting application. If this is the case, the user profile query continues directly and we already know that this query ends up in the success case. Otherwise, some further processing is required first.

Activity 3 (Check existence of Conditional Profile Subset for requesting application): We already know from activity 2 that there is at least a default profile subset. In this step, the User Profile Management Core checks whether the user's profile in addition contains at least one conditional profile subset.

Activity 4 (Identify required context types for Conditional Profile Subsets): In this step, we already know that there is at least one conditional profile subset. To be able to evaluate if the situational conditions of the conditional profile subsets match the user's current situation, the Context Client Module has to request the user's current context. For this purpose, the situational conditions of the conditional profile subsets are analysed with regard to the context types they depend on, so that only the context of those context types is requested by the Context Client Module. This activity step is delegated to the Context Client Module by the User Profile Management Core.

Activity 5 (Request user context of identified context types): Now, the Context Client Module requests the user context of the identified context types from the external context management framework.

Activity 6 (Find matching Conditional Profile Subset): This step is carried out by the User Profile Selection Module. The Context Client Module passes the received user context back to the User Profile Management Core, and the User Profile Management Core delegates the selection task to the User Profile Selection Module.

Activity 7 (Fetch matching Conditional Profile Subset): This case assumes that the User Profile Selection Module has identified a matching conditional profile subset in activity 6. This matching conditional profile subset is now fetched by the User Profile Management Core.

Activity 8 (Query Profile Subset): In this step, the user data within the resulting profile subset is queried. The resulting profile subset can either be the matching conditional profile subset from activity 7 or the default profile subset from activity 10. In our implementation, the query is expressed with the SPARQL query language [23], which is the standard RDF [17] query language from the World Wide Web Consortium (W3C).

Activity 9 (Return query result to requesting application): This is the final activity in the success case. In this step, the query result, i.e. the best matching user data, is returned to the requesting application.

Activity 10 (Fetch Default Profile Subset): In this step, the User Profile Management Core fetches the default profile subset. This is either because the default profile subset is the only profile subset or because no conditional profile subset matches the user's current situation.

Activity 11 (Return error message to requesting application): This is the final activity in the non-success case. In this step, the requesting application is notified that either there is no profile for the corresponding user, or there is no profile subset for the corresponding user, or no profile subset could be created for the requesting application.

Activity 12 (Check existence of Profile Subsets for other applications): In this step we already know that there is no profile subset for the requesting application. Hence, the only way to process the query successfully is to create a profile subset for this application based on existing profile subsets for other applications. Hence, in this step, the User Profile Management Core first has to check whether there are profile subsets in the user's profile for other applications. If this is the case, the query processing can continue, otherwise we end up in the non-success end point.

Activity 13 (Create Profile Subsets for requesting application): In this step, the User Profile Management core delegates the profile subset creation process to the User Profile Sharing Modules. The User Profile Sharing Module carries out the profile subset creation process based on existing profile subsets for other applications and based on existing schema mappings that are required for the transformation. If the creation process is successful, the query processing continues, otherwise we end up in the non-success end point.

An extract of the user profile management interface is shown in Appendix B: User Profile Management Interface. The appendix also includes documentation for users of the user profile management framework and developers of single modules. In the following we introduce the two user profile query methods, which could initiate the sequence of activities depicted in Figure 21:

getBestMatchingProfileSubset(in userID: String, in applicationID: String): ProfileSubset

Calling the *getBestMatchingProfileSubset* method, the best matching profile subset is returned as a whole, if existing. Thus, activity 8 of Figure 21 is skipped.

getBestMatchingUserData(in userID: String, in applicationID: String, in query: String): String

Calling the *getBestMatchingUserData* method represents the exact sequence of activities in Figure 21. The only difference to the *getBestMatchingProfileSubset* method is that the additional input parameter *query* is given, that activity 8 of Figure 21 is carried out, and that the returned result does not contain the matching profile subset as a whole but instead the query results as an XML encoded string.

```
PREFIX upr:<http://www.example.org/profile.owl#>
SELECT ?firstname ?email
WHERE
{ ?x upr:firstname ?firstname ;
      upr:email ?email .
}
```

Figure 22: SPARQL Query

```
<?xml version="1.0"?>
<sparql
        xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
        xmlns:xs="http://www.w3.org/2001/XMLSchema#"
        xmlns="http://www.w3.org/2005/sparql-results#" >
   <head>
           <variable name="firstname"/>
           <variable name="email"/>
   </head>
   <results ordered="false" distinct="false">
           <result>
                   <binding name="firstname">
                           <literal>Sandy</literal>
                   </binding>
                   <binding name="email">
                           <literal>Sandy.Smith@example.org</literal>
                   </binding>
           </result>
   </results>
</sparql>
```

Figure 23: SPARQL Query Result

Figure 22 shows an example of a SPARQL query. In this example the variables *firstname* and *email* are queried. Figure 23 depicts the query result if the SPARQL query of Figure 22 is applied to the example user model instance of Figure 12. The *head* element of the query result contains the queried variables *firstname* and *email*, the *results* element contains the value *Sandy* for variable *firstname* and the value *Sandy.Smith@example.org* for variable *email*.

3.2.2 Context Management Framework

Service platforms aim to take over potentially complex context discovery and processing steps from applications. For this reason, service platforms aim to provide distributed middleware architectures for the discovery and exchange of context information and a common context model. This context information can then be requested by applications or other platform services to provide context-aware features. Hence, application developers do not have to implement potentially complex context discovery and processing steps in an application-specific way within the applications. Instead, they can use the functionality provided by the service platform.

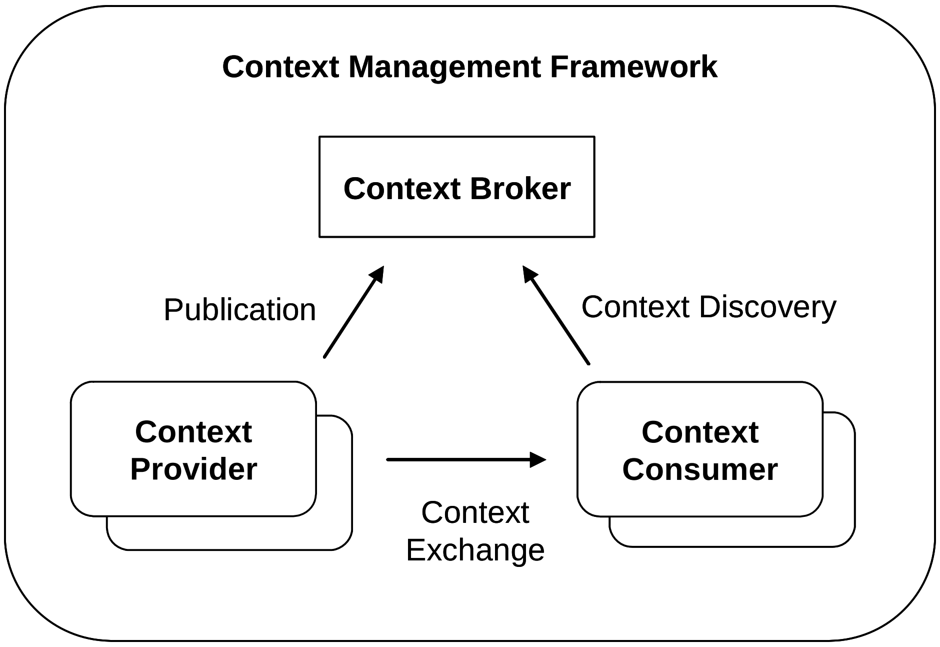


Figure 24: Context Management Framework

Such a context management framework is introduced in [86]. An enhancement of this framework is introduced in [87] and is called Knowledge Management Framework. Both versions follow the same concept, which us shown in Figure 24. Context Providers provide context information to Context Consumers. For this reason, Context Providers can register and publish their services and type of context they provide at the Context Broker. Context Consumers can contact the Context Broker for getting information on which Context Providers offer which kinds of context information. Afterwards, Context Consumers can contact and query the corresponding Context Providers directly. The communication between Context Consumer and Context Provider can follow both the request / response pattern and the subscription / notification pattern.

The user profile management framework as depicted in the section above is integrated into such a context management framework. In particular, it is integrated with the Knowledge Management Framework of [87]. There are two reasons for this integration. Firstly, the user profile management framework aims to provide the best matching user data concerning the current user's situation. For this purpose, information on the user's current situation is required. In particular, the User Profile Selection Module requires information about the current user's situation in order to select the best matching user data. The module, which is concerned with the user context request, is the Context Client Module of the user profile management framework. The Context Client Module is in fact a Context Consumer within the context management framework shown in Figure 24. Hence, the Context Client Module has to conform to the context management framework specific interfaces in order to discover, request and subscribe to context information. In particular, it also has to implement the context management framework specific call-back interface for receiving notifications by subscribed Context Providers.

The second reason for the integration is that the user profile management framework can take advantage of the subscription / notification pattern. As described in the section on fundamentals for user profile management 2.1.3, the user profile management framework in service platforms should follow the subscription / notification communication pattern. For this purpose, the user profile management framework conforms to the Context Provider specification. As a result, in addition to the implemented request / response interface described in section 3.2.1, it also implements the context management framework specific subscription interface. Hence, applications can subscribe to the user profile management framework. The advantage of such a subscription is that an application is automatically notified of changed user preferences as soon as the user profile management framework is notified of a changed user situation.

3.3 Summary

In the first part of this chapter we have introduced our user profile structure for service platforms. This user profile structure enables the specification of application-specific user sub-profiles as required in service platforms to enable different application-specific user attribute vocabularies. User attribute vocabularies are needed in order to give the user attributes a meaning and hence to make user attributes reusable by different applications. Different user attribute vocabularies need to be considered because different applications may need different user attributes or need extensions of available user attributes. For instance, this situation could occur in case a new application is deployed or in case an existing application is upgraded or extended with additional functionality. In both cases, additional user attributes may be required for these applications that are not yet covered by the vocabulary available within the service platform environment.

However, the user profile structure still allows using only one overall user sub-profile, should all application developers in a service platform agree on common semantics, i.e. on a common user attribute vocabulary. In this regard, our concept is very similar with the Generic User Profile (GUP) solution [36] [37] from the 3rd Generation Partnership Project (3GPP).

Besides application-specific user sub-profiles, the user profile structure also enables the definition of situation-specific sub-profiles by adding several situational conditions. These situational conditions have been designed with the aim to be easily understandable and editable for the normal non-technical user on the one hand, but still being expressive enough to enable useful and expressive conditions on the other hand. For this purpose, an approach has been chosen that is similar to the approach in the Gaia meta-operating system [65], which is similar to a simple clause in the English language of the form <subject> <verb> <object>. In combination with a user profile editor, the definition of such situational conditions could be carried out in just selecting items from select lists. That is, for each of the four attributes of a situational condition, the user does not have to fill out a blank text field, but can select from a pre-defined list. More on situational conditions is documented in chapter 5 together with the user profile selection mechanism.

Furthermore, we have also provided an analytic evaluation of runtime performance concerning two different search tasks for this user profile structure. The analytic evaluation compares the required comparison steps for finding matching situation-dependent user data between different user profile structures. The results show that clustering user profile data into several user sub-profiles, such as in our user profile structuring approach, is beneficial with regard to the runtime performance for search tasks.

In the second part of this chapter we have presented our user profile management framework, which is a flexible modular framework. It consists of three layers and several sub-modules, of which several sub-modules are easily exchangeable. As a result, the user profile management framework is easily adaptable to different requirements as could arise in different service platform environments. A detailed activity diagram has been provided that depicts the inter-working of sub-modules and explains the steps carried out during user profile request by clients of the user profile management framework.

In addition, the interaction with a context management framework has been documented. Due to the integration into this framework the user profile management system can offer subscription / notification functionality beside the request / response communication pattern.

The depicted user profile management framework and its interaction with and integration into a context management framework has been implemented within the SPICE (Service Platform for Innovative Communication Environment) project [4] [5] [6] and demonstrated at several project reviews.

4 User Profile and Context Ontology

In this chapter, we first describe our motivation for using ontologies in service platforms. Subsequently, we define our user profile ontology and our context ontology that are both part of the overall ontology model within the service platform.

Similar to the development of a software program by means of an object oriented programming language, there is no single correct methodology for the design of an ontology by means of an ontology language. In contrast to this, the development of a software program as well as the modelling of an ontology should be oriented to the goals the software program and the ontology strive for. However, in software programming, there are some programming conventions, e.g. for naming classes and methods. Similar to this, there are ontology modelling conventions. Noy and McGuinness [88], for example, present a guide to creating ontologies that include guidelines and conventions for the ontology modelling. This includes e.g. the definition of classes, class hierarchy and individuals. The forthcoming definition of our user profile ontology and context ontology also follows these modelling conventions.

4.1 Ontologies in Service Platforms

There are basically two reasons for the use of ontologies in service platforms. Firstly, the use of ontologies enables common semantics for different kinds of data throughout the whole service platform. Secondly, the use of ontologies enables the inference of additional information by means of ontology reasoning capabilities.

A service platform comprises multiple platform services as well as multiple end-user services. These services exchange different kinds of data, e.g. user profile data, user location data, service level agreements, device information etc. Hence, all these services should have a common understanding about the exchanged data in order to process it. Ontologies can provide an infrastructure to specify a vocabulary for arbitrary kind of data, and hence enable a common understanding of the related data. Furthermore, ontology languages such as the Web Ontology Language (OWL) [20] provide means to relate different ontologies, ontology classes, individuals etc. to each other. Different ontologies can be related to each other with the *owl:import* statement, ontology classes of different ontologies can be defined as equivalent with the *owl:equivalentClass* property, individuals of different ontologies can be defined identical with the *owl:sameAs* property, etc. This is also an important feature in case the service platform interacts with external parties, e.g. a third party service provider or another service platform in the domain of another operator. The latter one represents the roaming use case, in which a user leaves the domain of his home service platform operator and enters the domain of a foreign service platform operator. As different service platform operators and different service platforms usually adhere to different semantics for data, mappings between these different semantics are needed. Such mappings can be supported by ontology languages such as OWL with the above mentioned properties.

Within service platforms, there are various use cases for reasoning over data. For example, context management frameworks reason over low-level context information, i.e. sensor data such as GPS data, in order to derive high-level context information, i.e. meaningful data such as the room, in which the user is located. Another example could be the reasoning over content information in case the user requests a movie of a certain movie genre. In this case, classification information within ontologies could be applied to find all movies of sub-categories of a certain movie genre. Furthermore, reasoning could be used for learning user preferences from usage histories. Last but not least, reasoning could be used to find matching user data concerning the user’s current situation in order to personalise services and the user’s environment. The latter one is the main issue of this thesis and is explained in more detail in chapter 5. The required reasoning capability are enabled by ontology languages such as OWL, in particular OWL DL (OWL Description Logics), which is an OWL sub-language that provides computational completeness (all entailments are guaranteed to be computed) and decidability (all computations will finish in finite time).

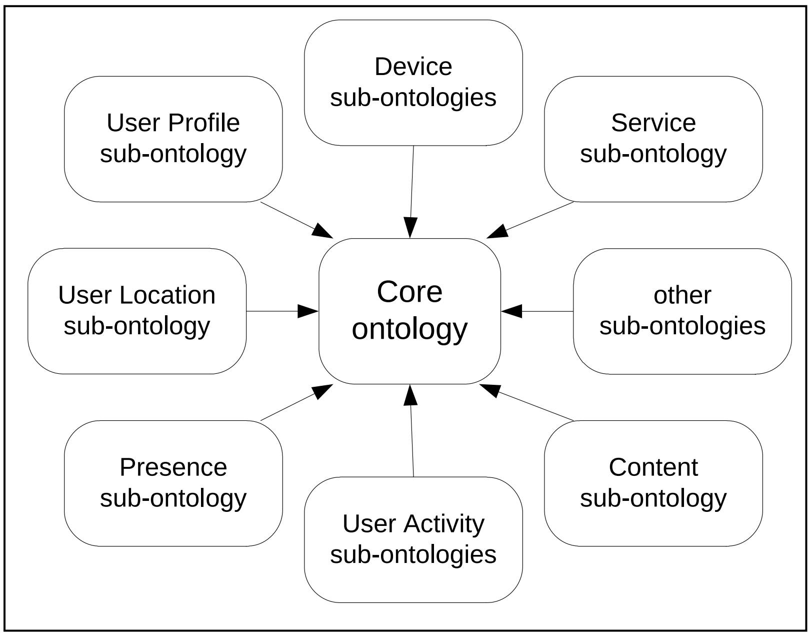


Figure 25: Ontology Infrastructure in Service Platforms

Figure 25 shows how different kinds of vocabularies, i.e. ontologies, could be organised in a service platform. The core ontology should include concepts and properties that are relevant for all kinds of vocabularies. Specific kinds of data should in addition be defined in sub-ontologies. For example, a concept for service platform users is required throughout the service platform for many cases. One might like to exchange the information that a user is located in a particular building (user location sub-ontology), one might like to exchange the information that a user owns a particular device (device sub-ontology), or one might like to exchange the information that a user is consuming certain content (content sub-ontology). Hence, the user concept should be specified within the core ontology instead of specifying three different concepts for the same thing within the sub-ontologies. Sub-ontologies should then refer to these concepts if needed. Sub-ontologies can also refer to concepts of other sub-ontologies if needed. One of the advantage of having a core ontology and different sub-ontologies compared to having a single overall ontology is that we only have to load those parts, which are needed for the processing of data. As a result, we do not have to load a potentially huge amount of concept and property definitions a single overall ontology would include.

The ontologies presented next in this chapter are our user profile ontology and our context ontology. Both ontologies could be described as sub-ontologies within the ontology infrastructure of a service platform.

4.2 User Profile Ontology

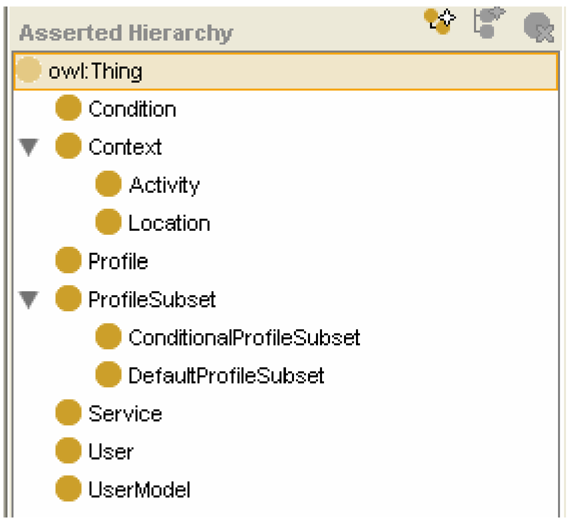
In this section, we describe our user profile ontology. We named this ontology UPOS (User Profile Ontology with Situation-Dependent Preferences Support). UPOS provides a schema for a user profile structure. It does not specify any user attributes. However, as described later, individual and, hence, application-specific user attribute schemas can extend UPOS. In

UPOS, user profiles can consist of several application-specific sub-profiles. Optionally, these sub-profiles can be conditional, i.e. they can depend on situational conditions such as a user location or user activity. In doing so, UPOS follows the same goal as the user profile structure presented in section 3.1.1. However, whereas the user profile structure has been presented by means of a class diagram, UPOS provides the corresponding ontology specification.

UPOS has been specified with OWL [20], in particular the specification is in the sub-language OWL DL, which provides computational completeness and decidability. The complete specification is shown in Appendix C: Specification of the User Profile Ontology. In the following, we present the ontology by means of visualisations, and explain classes, properties and extensions. As mentioned above, the user profile ontology is a sub-ontology in the service platform ontology infrastructure. However, for the clearness of presentation, we introduce all classes and properties as if they were part of the user profile ontology. Some classes and properties could in fact be part of the ontology core of the service platform ontology infrastructure and hence be imported instead.

4.2.1 Classes and Properties

Figure 26 shows the class hierarchy of the user profile ontology. The introduced classes are mostly the same as introduced in the user profile structure in section 3.1.1. In addition, context related high-level classes are introduced in order to express situational conditions with context parameters.


Asserted Hierarchy

- owl:Thing
  - Condition
  - Context
    - Activity
    - Location
  - Profile
  - ProfileSubset
    - ConditionalProfileSubset
    - DefaultProfileSubset
  - Service
  - User
  - UserModel

Figure 26: Class Hierarchy of the User Profile Ontology

The defined classes are as follows:

- User: A user of the service platform.
- Profile: A user profile.
- ProfileSubset: A service-specific and optionally situation-specific subset of a user profile. DefaultProfileSubset and ConditionalProfileSubset are specialisations of ProfileSubset.
- DefaultProfileSubset: A service-specific subset of a user profile that includes a default user model. Related to each service, only one default profile subset is allowed.
- ConditionalProfileSubset: A service and situation-specific subset of a user profile that includes a user model for a specific situation.
- UserModel: A user model.
- Service: A service associated with the service platform.

- Condition: A condition specifies the situation, in which a conditional profile subset is valid, e.g. the home or the office location.
- Context: A context, e.g. a user activity or a user location. Location and Activity are specialisations of Context.
- Location: A user location.
- Activity: A user activity.

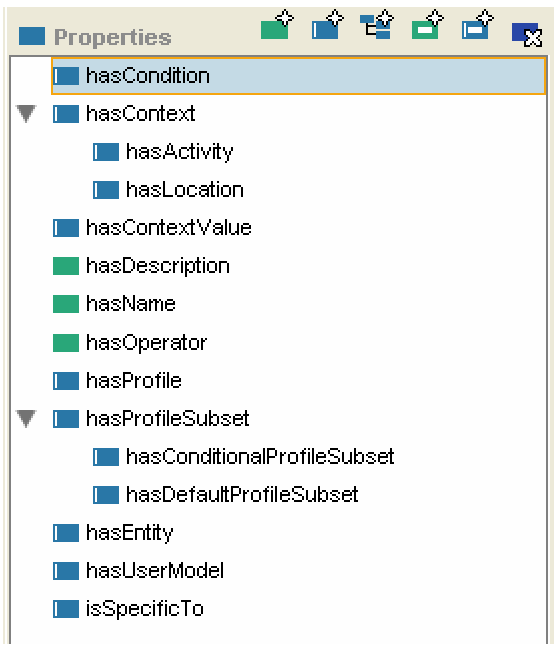


Figure 27: Properties of the User Profile Ontology

Figure 27 shows the defined properties. Object properties, i.e. relations between instances of classes, are coloured blue, whereas datatype properties, i.e. relations between instances of classes and RDF literals and XML Schema datatypes, are coloured green. The defined object properties are as follows:

- hasProfile: A user has a profile.
- hasProfileSubset: A user profile has a profile subset. HasDefaultProfileSubset and hasConditionalProfileSubset are specialisations of hasProfileSubset.
- hasDefaultProfileSubset: A user profile has a default profile subset. Related to each service, only one default profile subset is allowed.
- hasConditionalProfileSubset: A user profile has a conditional profile subset.
- hasUserModel: A profile subset has a user model.
- isSpecificTo: A profile subset is specific to a service.
- hasCondition: A conditional profile subset has a condition.
- hasEntity: A condition is linked to an entity, e.g. a user or a room.
- hasContextValue: A condition has a context value, e.g. a certain user location.
- hasContext: A user is in a certain context. HasLocation and hasActivity are specialisations of hasContext.
- hasLocation: A user has a location.
- hasActivity: A user is engaged in an activity.

The defined datatype properties are as follows:

- hasName: A profile subset has a name. This name has to be unique within a user profile.
- hasDescription: A profile subset has an optional description.

- hasOperator: A condition has an operator, e.g. equal, notEqual, greaterThan, greaterThanOrEqual, lessThan or lessThanOrEqual.

4.2.2 Ontology Visualisation

Figure 28 shows the defined classes and the relations between those. However, not all details are shown in this figure. For example, cardinality restrictions are not shown in this figure. For this purpose, please refer to the complete specification in Appendix C: Specification of the User Profile Ontology.

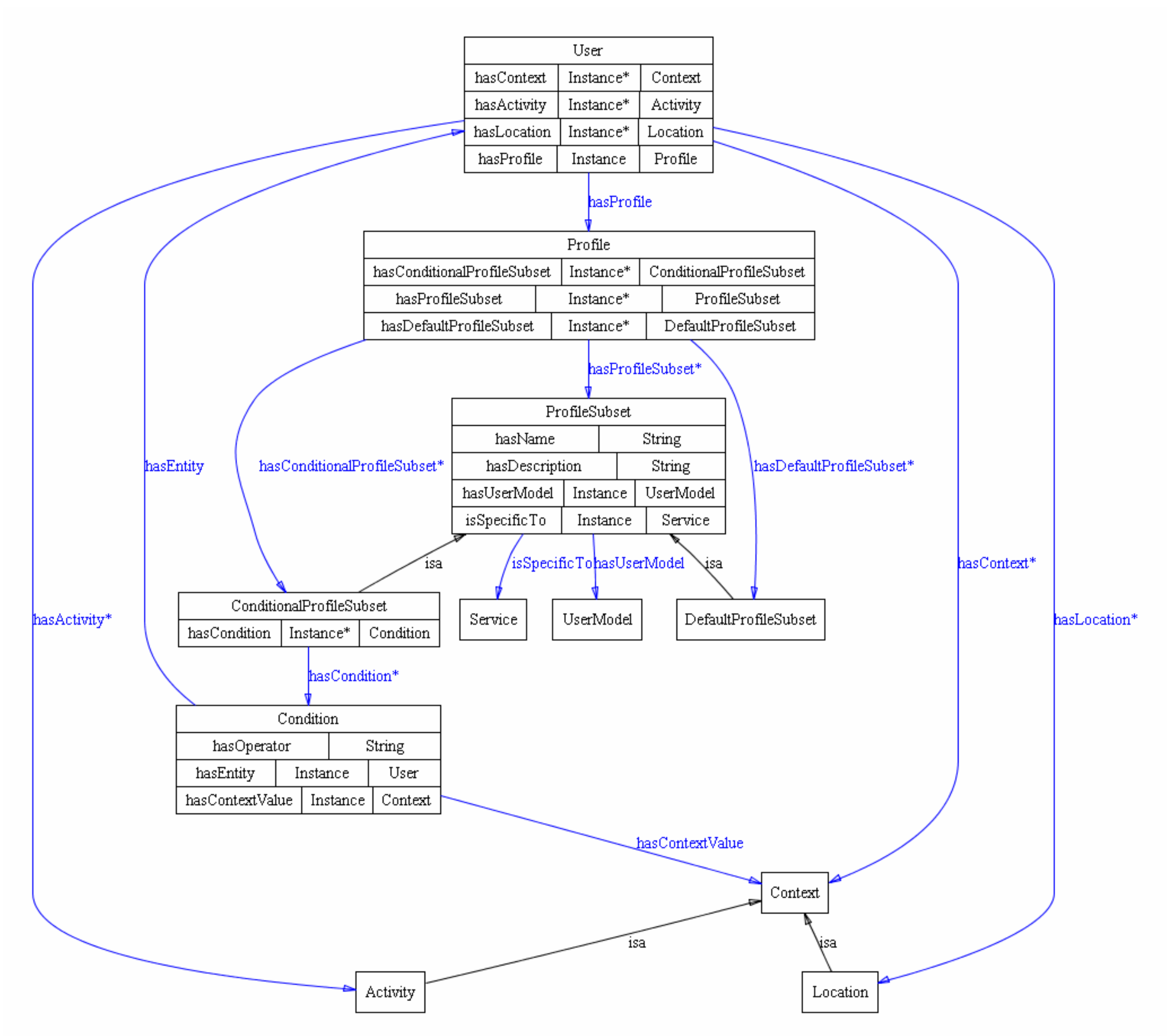

User
hasContext | Instance* | Context
hasActivity | Instance* | Activity
hasLocation | Instance* | Location
hasProfile | Instance | Profile
hasProfile
Profile
hasConditionalProfileSubset | Instance* | ConditionalProfileSubset
hasProfileSubset | Instance* | ProfileSubset
hasDefaultProfileSubset | Instance* | DefaultProfileSubset
hasProfileSubset*
ProfileSubset
hasName | String
hasDescription | String
hasUserModel | Instance | UserModel
isSpecificTo | Instance | Service
hasEntity
hasConditionalProfileSubset*
hasDefaultProfileSubset*
hasContext*
hasActivity*
hasLocation*
isa
isSpecificTo
hasUserModel
isa
ConditionalProfileSubset
hasCondition | Instance* | Condition
Service
UserModel
DefaultProfileSubset
hasCondition*
Condition
hasOperator | String
hasEntity | Instance | User
hasContextValue | Instance | Context
hasContextValue
Context
isa
isa
Activity
Location

Figure 28: Relations between Classes of the User Profile Ontology

The boxes in Figure 28 represent classes, with the class name in the first row of a box. The subsequent rows within a box represent the properties of a class, also called slots. The User class for example has the properties hasContext, hasActivity, hasLocation and hasProfile. The arrows between boxes represent object properties. The asterisk of object properties denotes that an instance of a class can have multiple relationships of this kind of property to other instances. For example, whereas a User can have only one Profile, expressed by the hasProfile property that is shown without an asterisk, a user can have multiple Contexts, expressed by the hasContext property that is shown with an asterisk.

In comparison to the condition class defined in the user profile structure in section 3.1.1, the condition class in the ontology does not contain an attribute for the type of context the hasContextValue refers to. This is not required in the ontology definition for two reasons. On the one hand, we can infer the type of context, e.g. user location or user activity, from the class hierarchy of the context ontology applied on the hasContextValue attribute. On the other hand, if we assume that there is a sub-ontology for each type of context, then the URI of the sub-ontology is sufficient to identify the type of context.

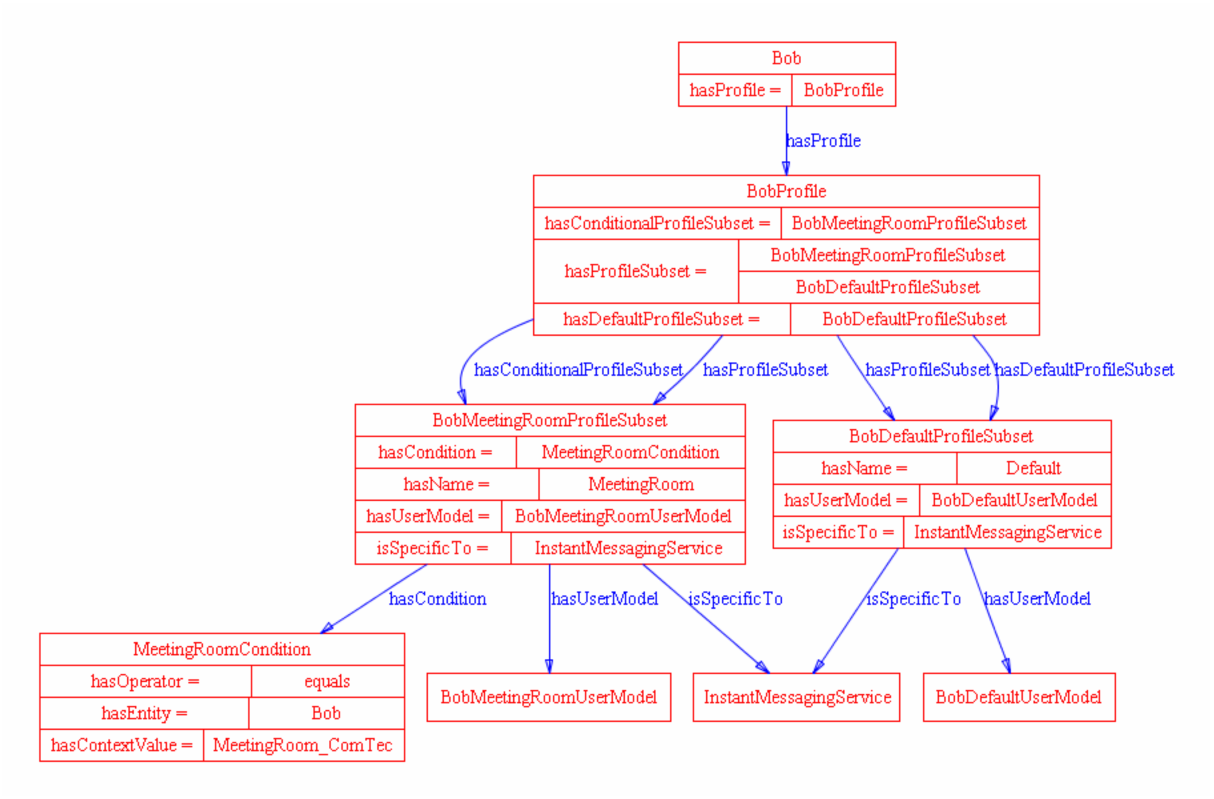


Figure 29: User Profile Instance Example

Figure 29 shows an instance example. User Bob has a profile called BobProfile. This profile includes two profile subsets, the default profile subset BobDefaultProfileSubset and the conditional profile subset BobMeetingRoomProfileSubset. Both are related to the service InstantMessagingService. Whereas the default profile subset contains the user model BobDefaultUserModel, the conditional profile subset contains the user model BobMeetingRoomUserModel and the situational condition MeetingRoomCondition.

4.2.3 Extensibility

As mentioned above, UPOS specifies a vocabulary for a user profile structure, but no user attribute vocabulary. This is because the user profile structure is unique within the whole service platform, whereas user attribute vocabularies have to be extensible to serve the requirements of arbitrary services, and hence may be service-specific.

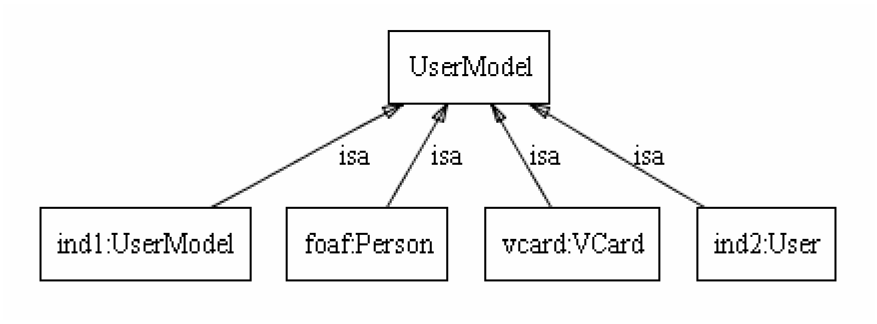


Figure 30: Specialisations of the UserModel Class

Figure 30 shows how UPOS and individual user attribute ontologies are connected. The UserModel class is defined in UPOS and aims at consisting the actual user attributes. These user attributes can now be defined by individual user attribute ontologies. For example, the FOAF schema [41] and the vCard schema [42], both already introduced in section 2.2.1.1, provide vocabularies for user attributes. If using FOAF, user attributes are part of the top-level class foaf:Person, if using vCard, user attributes are part of the top-level class vcard:VCard. The connection between these top-level classes and UPOS can be done by defining those top-level classes as specialisations of the UserModel class. Figure 30 depicts this specialisation. Besides FOAF and vCard arbitrary other individual user attribute schemas can be used as also shown in the figure. Ind1:UserModel represents the UserModel class of an individual schema called ind1 (individual 1), and ind2:User represents the User class of an individual schema called ind2 (individual 2).

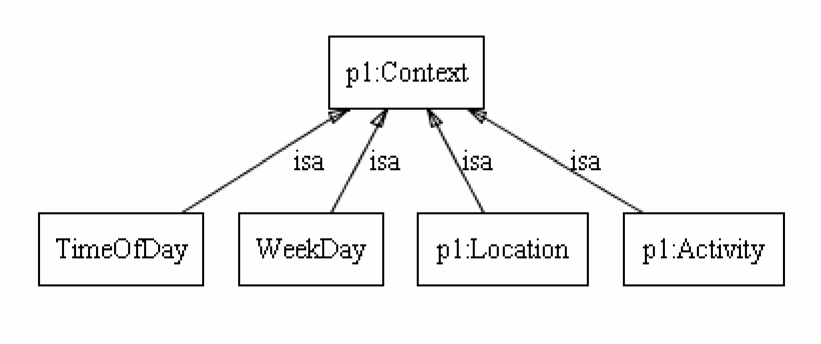


Figure 31: Specialisations of the Context Class

Similar to the above shown specialisation of the UserModel class, UPOS can also be extended with concrete context ontologies. Figure 31 shows this specialisation, in which the prefix p1 represents the UPOS namespace. On the one hand, the existing classes Location and Activity can be inherited to include specific vocabularies for user location and user activity. On the other hand, additional context types, which are not specified in UPOS such as time of day and weekday, as well as the related ontologies, can be added.

4.3 Context Ontology

In this section, we describe how to design context ontologies in such a way that we can develop an advanced user profile selection mechanism that is based on ontology reasoning. The approach we follow is not the development of a user profile selection mechanism based on arbitrary context ontologies. Instead, both development steps are linked together. The development of context ontologies is based on the goals we strive with our user profile selection mechanism. Hence, we first decide on the use cases for user profile selection and then develop the context ontologies in such a way that these use cases can be realised.

In the following subsections, we show the development for one such example context ontology, particularly a user location ontology. We only focus on user locations. However, the same modelling approach could be used for other kinds of context ontologies, such as user activity ontologies. We first show the modelling goals for the user location ontology. Afterwards, we show the actual realisation to reach these goals.

4.3.1 Modelling Goal for the Location Ontology

The user profile selection mechanism aims at selecting the matching sub-profile, i.e. the sub-profile that matches the user's current situation. For this purpose, the user profile selection mechanism evaluates the situational conditions that are attached to conditional sub-profiles. Examples for such situational conditions have been presented in Table 1, where the operator attribute of situational conditions depends on the type of context that is expressed by the situational condition. As explained in section 3.1.1, the structure of situational conditions aims to be not only very expressive, but also easily understandable by non-technical users. For this purpose, the meaning of the situational conditions has to be easily understandable. In the following, we only concentrate on operators for the context type location. We now define example operators, which aim to be expressive and which aim to be easily understandable by non-technical users. Our example operators for the context type location are as follows:

1. equals
2. notEquals
3. isWithin
4. notIsWithin
5. isConnectedTo
6. notIsConnectedTo
7. isA
8. notIsA
9. isWithinA
10. notIsWithinA
11. isConnectedToA
12. notIsConnectedToA

In the following, we first depict one example situational condition for each of the depicted operators in Table 4. Afterward, the operators are explained.

Table 4: Situational Conditions with Context Type Location

EntityID	*ContextType*	*Operator*	*ContextValue*
Bob	location	equals	MeetingRoom-1
Bob	location	notEquals	Office-2
Bob	location	isWithin	OfficeBuilding-3
Bob	location	notIsWithin	Premises-A
Bob	location	isConnectedTo	PrinterSpace-3
Bob	location	notIsConnectedTo	PrinterSpace-2
Bob	location	isA	MeetingRoom
Bob	location	notIsA	VideoProjectorSpace
Bob	location	isWithinA	UmtsAccessRange
Bob	location	notIsWithinA	WlanAccessRange

Bob	location	isConnectedToA	PrinterSpace
Bob	location	notIsConnectedToA	PrinterSpace

The operators and examples in Table 4 are explained in the following. However, the depicted operators should be considered as examples in order to explain our ontology modelling approach. Thus, if a developer wants to define other operators for the context type location, i.e. other relationships between locations, then this can be done of course, but usually leads to a different ontology at the end.

equals
The equals operator is an instance related operator. This means that it is only used in combination with context values that are instances of ontology concepts, e.g. the room instance MeetingRoom-1. The equals statement is true, if and only if the context value of the current user's situation represents the same location instance as the context value of the situational condition. A user may e.g. specify such a situational condition for the personalisation of applications in case she is in that meeting room. For instance, the presentation and communication settings of the laptop could be adapted to the video projector and network capabilities of MeetingRoom-1.

notEquals
The notEquals operator is interpreted as the opposite of the equals operator.

isWithin
The isWithin operator is an instance related operator, such as the equals operator. The isWithin statement is true, if and only if the context value of the current user's situation represents a location instance that is located within the location instance represented by the context value of the situational condition, e.g. within the office building instance OfficeBuilding-3. This example could e.g. be used to personalise communication setting within a specified location such as the premises the user works, a UMTS access range of a certain network operator, or even a whole spatial region.

notIsWithin
The notIsWithin operator is interpreted as the opposite of the isWithin operator.

isConnectedTo
The isConnectedTo operator is again an instance related operator, such as the equals operator. The isConnectedTo statement is true, if and only if the context value of the current user's situation represents a room instance that is connected to the room instance represented by the context value of the situational condition, e.g. the room instance PrinterSpace-3. This example could e.g. be used to personalise printer jobs in case one is spatially connected to a printer space, i.e. a room with a certain printer.

notIsConnectedTo
The notIsConnectedTo operator is interpreted as the opposite of the isConnectedTo operator.

isA
The isA operator is a concept related operator. That is, it is only used in combination with context values that are ontology concepts instead of ontology instances, e.g. the concept MeetingRoom, OfficeBuilding, etc. The isA statement is true, if and only if the context value of the current user's situation represents an instance of the location concept represented by the context value of the situational condition. This example could be used in a similar way as the

equals operator, but related to a specified location concept instead of a specified location instance. One such example situational condition could be used to personalise communication applications such as telephony, news, messages and firewall in case the user is in a certain type of location, e.g. a meeting room or a laboratory.

notIsA
The notIsA operator is interpreted as the opposite of the isA operator.

isWithinA
The isWithinA operator is a concept related operator such as the isA operator. The isWithinA statement is true, if and only if the context value of the current user's situation represents a location instance that is located within a location instance of the location concept represented by the context value of the situational condition, e.g. the concept UmtsAccessRange. This example could be used in a similar way as the isWithin operator, but related to a specified location concept instead of a specified location instance. For example, it could be used to personalise communication setting within a specific type of access range, e.g. a UMTS access range for fast Internet access.

notIsWithinA
The notIsWithinA operator is interpreted as the opposite of the isWithinA operator.

isConnectedToA
The isConnectedToA operator is again a concept related operator such as the isA operator. The isConnectedToA statement is true, if and only if the context value of the current user's situation represents a room instance that is connected to a room instance of the room concept represented by the context value of the situational condition, e.g. the concept PrinterSpace. This example could be used in a similar way as the isConnectedTo operator, but related to a specified room concept instead of a specified individual room.

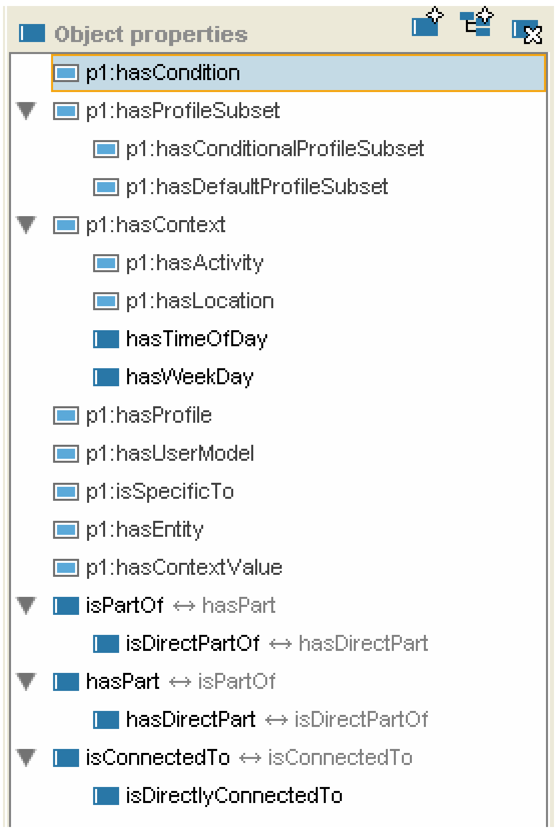
notIsConnectedToA
The notIsConnectedToA operator is interpreted as the opposite of the isConnectedToA operator.

It should be mentioned that there is a challenge concerning some of the introduced example operators in combination with the user profile ontology as defined above. The example operators isA, notIsA, isWithinA, notIsWithinA, isConnectedToA and notIsConnectedToA are operators that have a concept as context value instead of a concrete individual. Expressing such situational conditions in the above defined user profile ontology is not allowed as the context value of the situational condition is specified to be an individual. However, this issue could be addressed with a corresponding change of the user profile ontology, which would allow the context value to be a concept or an individual. Unfortunately, this change would make the user profile ontology be in the OWL Full sub-language of OWL [20] instead of the OWL DL sub-language. Hence, we would leave the computationally complete and decidable part of OWL. However, this would not be disadvantageous for our user profile selection mechanism, as we do not intend to carry out ontology reasoning over the user profile ontology, but over the location ontology. The corresponding change of the hasContextValue property of the user profile ontology is shown in Appendix D: Additional Property Specification.

4.3.2 Location Classes and Properties

In the above section, we have introduced our example operators for the context type location. In this section, we now explain how the location ontology is designed so that these operators can be realised. For this purpose, we describe class and property definitions. However, not all details of the specification are shown in this section. The complete formal specification is given in Appendix E: Specification of the Location Ontology.

As for the user profile ontology, the location ontology has also been specified with the OWL DL sublanguage of OWL [20], which provides computational completeness and decidability. These are two important features, as the user profile selection mechanism applies ontology reasoning over the location ontology, and as we want to ensure that the user profile selection mechanisms always provides a result. The user profile selection mechanism is described in detail in chapter 5.


Object properties

- p1:hasCondition
- p1:hasProfileSubset
  - p1:hasConditionalProfileSubset
  - p1:hasDefaultProfileSubset
- p1:hasContext
  - p1:hasActivity
  - p1:hasLocation
  - hasTimeOfDay
  - hasWeekDay
- p1:hasProfile
- p1:hasUserModel
- p1:isSpecificTo
- p1:hasEntity
- p1:hasContextValue
- isPartOf ↔ hasPart
  - isDirectPartOf ↔ hasDirectPart
- hasPart ↔ isPartOf
  - hasDirectPart ↔ isDirectPartOf
- isConnectedTo ↔ isConnectedTo
  - isDirectlyConnectedTo

Figure 32: Properties of the Location Ontology

Figure 32 shows the object properties of the location ontology together with the object properties of the user profile ontology. The object properties of the user profile ontology have already been introduced in section 4.2.1 and can be identified with the prefix p1. They are still shown in Figure 32, because the location ontology imports the user profile ontology in order to reuse classes already specified in the user profile ontology. The newly introduced object properties of the location ontology are described in the following:

- isPartOf: A location is part of another location. This property is defined as transitive. It is also defined as the inverse of the hasPart property, which is depicted by means of the arrow with the two arrowheads in Figure 32. The isPartOf property is used to realise the isWithin, notIsWithin, isWithinA and notIsWithinA operators.
- isDirectPartOf: A location is direct part of another location. IsDirectPartOf is a specialisation of isPartOf. It is also defined as the inverse of the hasDirectPart property. The isDirectPartOf property is also used to realise the isWithin, notIsWithin, isWithinA and notIsWithinA operators.

- hasPart: A location has another location as a part. This property is defined as transitive. It is also defined as the inverse of the isPartOf property. The hasPart property is also used to realise the isWithin, notIsWithin, isWithinA and notIsWithinA operators.
- hasDirectPart: A location has another location as a direct part. HasDirectPart is a specialisation of hasPart. It is also defined as the inverse of the isDirectPartOf property. The hasDirectPart property is also used to realise the isWithin, notIsWithin, isWithinA and notIsWithinA operators.
- isConnectedTo: A room is connected to another room. This property is defined as symmetric and transitive. The isConnectedTo property is used to realise the isConnectedTo, notIsConnectedTo, isConnectedToA and notIsConnectedToA operators.
- isDirectlyConnectedTo: A room is directly connected to another room. This property is defined as symmetric. IsDirectlyConnectedTo is a specialisation of isConnectedTo. The isDirectlyConnectedTo property is also used to realise the isConnectedTo, notIsConnectedTo, isConnectedToA and notIsConnectedToA operators.
- hasTimeOfDay: A user is related to a time of day, i.e. the time of day a user is situated in. HasTimeOfDay is a specialisation of the hasContext property of the user profile ontology.
- hasWeekDay: A user is related to a weekday, i.e. the weekday a user is situated in. HasWeekDay is a specialisation of the hasContext property of the user profile ontology.

So far, we have not created and added any individuals and relationships between individuals to the location ontology. However, relationships between individuals, such as individual rooms, office buildings or premises are always expressed with the isDirectPartOf, hasDirectPart or isDirectlyConnectedTo properties. That is, if we consider a location consisting of different smaller location entities, then relationships between these location entities are always defined in such a way that a location entity is direct part of the next bigger location entity. For example, if we consider an office building consisting of the entities floors and rooms, then relationships between the building, its floors and rooms are always defined in such a way that rooms are direct parts of the next bigger entity, i.e. floors, and floors are direct parts of the next bigger entity, i.e. buildings, instead of defining rooms as parts of buildings. Hence, relationships between individuals are not expressed with the isPartOf, hasPart and isConnectedTo properties. These properties are only used within the ontology reasoning step, in which inference over transitivity and symmetry and other ontology language constructs is carried out. The reasoning step is explained later in chapter 5, also showing ontology reasoning examples.

The reason for distinguishing between the above definitions of isDirectPartOf and isPartOf, hasDirectPart and hasPart, and isDirectlyConnectedTo and isConnectedTo is that the ontology is easier to administrate in case new individuals are added, or existing individuals are removed or changed at a later stage. Just consider to add a new individual of type building to the ontology. Then all relationships that have to be added to the ontology are the single relationship that this building is direct part of the corresponding street. Everything else, i.e. the information that the building is part of a quarter, is part of a city, is part of a region, is part of a country and is part of a continent can be inferred over the transitive isPartOf property from existing relationships in the ontology. In comparison to this, without distinguishing between the definitions of isDirectPartOf, isPartOf and the related creation rule for individuals, each of the above relationships would have to be added separately, which is a time-consuming and erroneous procedure.

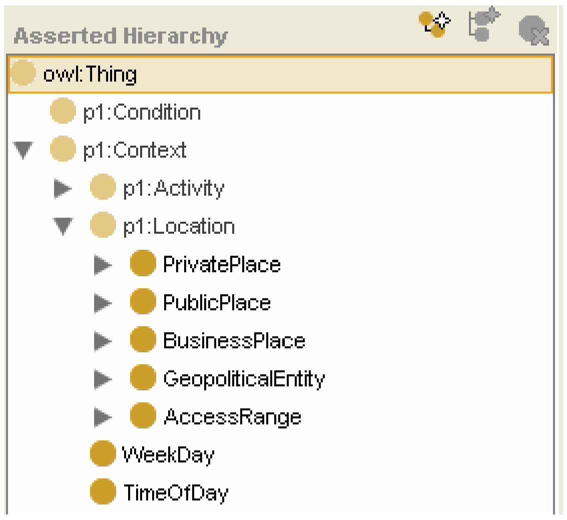
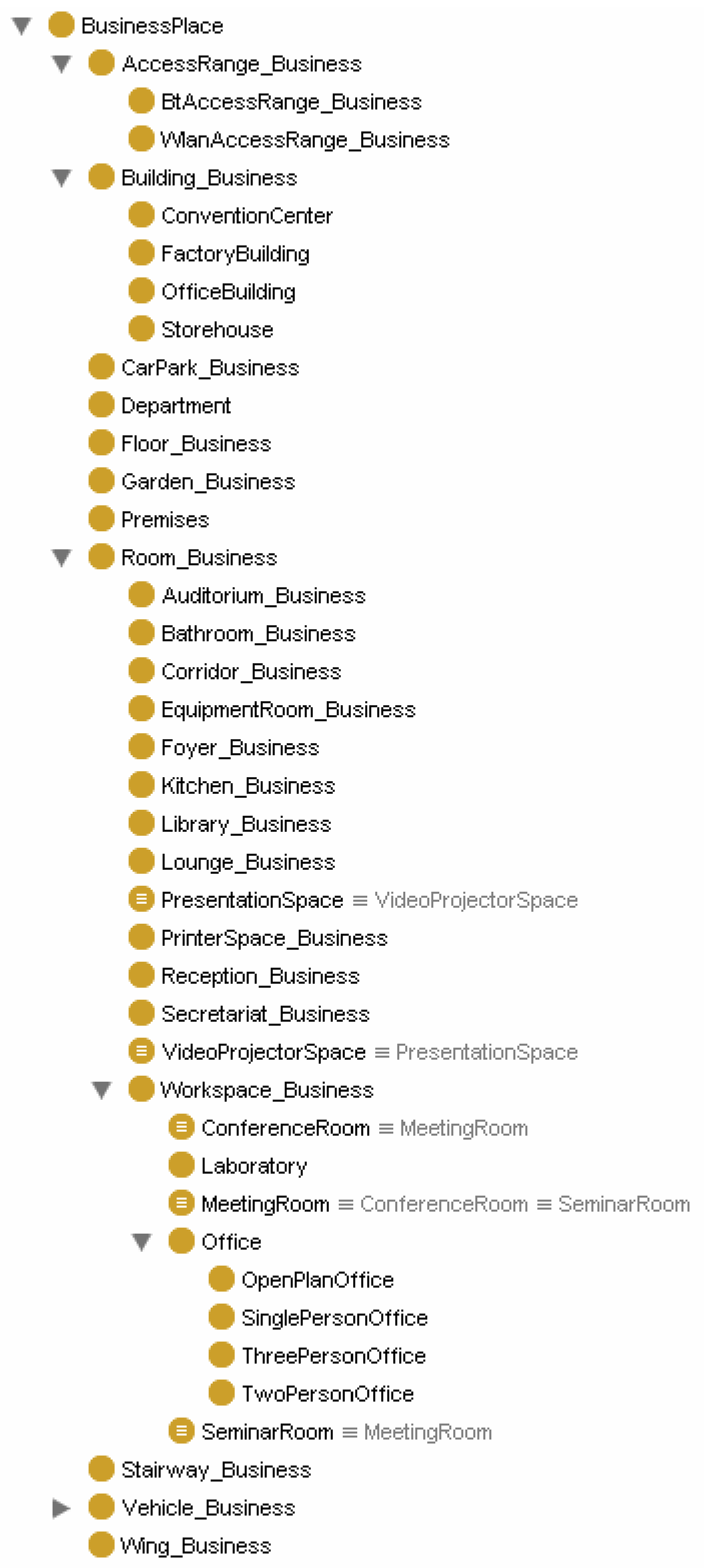


Figure 33: Top-Level Class Hierarchy of the Location Ontology

As mentioned in section 2.2.3.2, our location ontology should not only represent location concepts from a pure spatial point of view. Instead, location concepts should also be modelled according to the function of individual spaces. Consequently, rooms should be modelled as meeting rooms, offices, printer spaces, presentation spaces, etc., and buildings should be modelled as office buildings, factory buildings, private homes, etc. In addition, space concepts should be distinguished between whether they are in a private, business or public environment, because the user's behaviour differs on whether she is in a private, business or public space, and whether a room is an office, a meeting room, a laboratory or a lounge.

Figure 33 shows the top-level specialisations PrivatePlace, PublicPlace, BusinessPlace, GeopoliticalEntity and AccessRange of the Location class as the result of this approach. The figure also shows some classes of the user profile ontology, which have already been introduced in section 4.2.1 and can be identified with the prefix p1. They are still shown in Figure 33 because the classes specified in the location ontology are specialisations of the classes specified in the user profile ontology.

The PrivatePlace concept is inherited by location concepts addressing private places of a user such as private rooms, homes, and private WLAN access ranges. The PublicPlace concept is inherited by location concepts that describe public places such as tourist places, airport lounges, cafes, hotels, shopping centres and places for entertainment. The BusinessPlace concept on the other hand is inherited by location concepts describing a business place, such as departments, office buildings, offices, meeting rooms and business related WLAN access ranges. The GeopoliticalEntity concept has the specialisations City, Region, Country and Continent. Finally, the AccessRange concept models GSM and UMTS based access ranges.



- BusinessPlace
  - AccessRange_Business
    - BtAccessRange_Business
    - WlanAccessRange_Business
  - Building_Business
    - ConventionCenter
    - FactoryBuilding
    - OfficeBuilding
    - Storehouse
  - CarPark_Business
  - Department
  - Floor_Business
  - Garden_Business
  - Premises
  - Room_Business
    - Auditorium_Business
    - Bathroom_Business
    - Corridor_Business
    - EquipmentRoom_Business
    - Foyer_Business
    - Kitchen_Business
    - Library_Business
    - Lounge_Business
    - PresentationSpace ≡ VideoProjectorSpace
    - PrinterSpace_Business
    - Reception_Business
    - Secretariat_Business
    - VideoProjectorSpace ≡ PresentationSpace
    - Workspace_Business
      - ConferenceRoom ≡ MeetingRoom
      - Laboratory
      - MeetingRoom ≡ ConferenceRoom ≡ SeminarRoom
      - Office
        - OpenPlanOffice
        - SinglePersonOffice
        - ThreePersonOffice
        - TwoPersonOffice
      - SeminarRoom ≡ MeetingRoom
  - Stairway_Business
  - Vehicle_Business
  - Wing_Business

Figure 34: Specialisation of the BusinessPlace Concept of the Location Ontology

Figure 34 shows the complete sub-hierarchy of the BusinessPlace concept as part of the location hierarchy. This sub-hierarchy models buildings, parts of buildings and rooms based on its function, such as a Laboratory, a Lounge_Business or a MeetingRoom. As already mentioned in section 2.2.3.2, deriving indoor user locations down to a single room is feasible [80] [81]. We also enabled the definition of different room types that represent the same functionality. The MeetingRoom class, for instance, is defined as equivalent class to the ConferenceRoom class as can be seen in Figure 34. This definition is done with the owl:equivalentClass property. Both modelling constructs, class hierarchy as well as owl:equivalentClass, are used to realise the equals, notEquals, isA and notIsA operators. Detailed reasoning examples concerning all the operators are explained later in section 5.2. The specialisations of the PrivatePlace and PublicPlace concepts are not shown here. However, they are modelled in the same way. The corresponding formal specification with all details is given in Appendix E: Specification of the Location Ontology.

4.4 Summary

In the first part of this chapter, we have described the need to use ontologies in service platforms and explained a potential approach for an ontology infrastructure for this environment. There are two main reasons for the use of ontologies in service platforms. Firstly, the use of ontologies enables common semantics for different kinds of data throughout the whole service platform. This is particularly important for our user profile ontology. Secondly, the use of ontologies enables the inference of additional information by means of ontology reasoning capabilities. This is in particular important for our context ontology in combination with our user profile selection mechanism that is described in chapter 5.

In the second part of this chapter we have introduced our user profile ontology as a sub-ontology within the ontology infrastructure of a service platform. The specification of the user profile ontology is based on the user profile structure introduced in chapter 3. Besides a description of ontology classes and properties, we have also provided visualisations of the ontology, and a discussion on how the ontology can be extended with individual user attribute vocabularies such as FOAF [41] and vCard [42] on the one hand, and with individual context ontologies on the other hand. The complete formal specification of the user profile ontology is provided in Appendix C: Specification of the User Profile Ontology.

In the third part of this chapter our location ontology has been presented as an example of a context ontology for service platforms. However, the same modelling approach as shown in this section could also be used for other kinds of context ontologies, such as user activity ontologies. We have described how to design context ontologies in such a way that we can develop an advanced user profile selection mechanism that is based on ontology reasoning. The approach we have followed is not the development of a user profile selection mechanism based on arbitrary context ontologies. Instead, in the approach we have taken, both development steps have been linked together. That is, the development of the example location ontology is based on the goals we strive with our user profile selection mechanism.

The modelling goal for the location ontology is connected to the example operators for expressing situational conditions. In the first step, we have identified and described useful example operators. Afterwards, we have shown the actual realisation to reach these goals, which led to the specification of our location ontology. The location ontology has been presented by means of class and property descriptions, as well as visualisations and descriptions of the class hierarchy. Again, the complete formal specification of the location ontology is provided in Appendix E: Specification of the Location Ontology.

5 Automatic Selection of User Profiles

In this chapter we present the User Profile Selection Module that has already been introduced briefly in section 3.2.1 as part of the overall user profile management framework. On the one hand, it is described by means of its interaction with external components. On the other hand, we show details of the internal ontology reasoning step, i.e. the reasoning over the location ontology introduced in section 4.3, that takes place within the user profile selection process. In doing so, we show reasoning examples for all the location-related operators introduced in section 4.3, i.e. for various different situational conditions.

5.1 User Profile Selection Module

The User Profile Selection Module is the module that carries out the selection of the matching profile subset for the user's current situation. For this purpose, it comprises an evaluation functionality that applies ontology reasoning in the selection process. This evaluation functionality is the central part of this thesis and is explained in the following: First, the general functioning of the module is explained by means of the included processing steps. Afterwards, a detailed activity diagram is provided that depicts the actual algorithm that is used for the user profile selection process. Finally, we show various examples of the ontology reasoning step carried out within the User Profile Selection Module for different situational conditions, i.e. different operators.

5.1.1 Functioning of the User Profile Selection Module

The user profile selection process can be described by means of two processing steps, the query creation process and the condition evaluation process, see Figure 35. In the query creation process, the actual query has to be prepared. Queries are implemented as SPARQL queries [23], the standard query language for RDF [17] from the W3C. An example of a SPARQL query has already been shown in Figure 22 and the corresponding query result in Figure 23 in section 3.2.1.

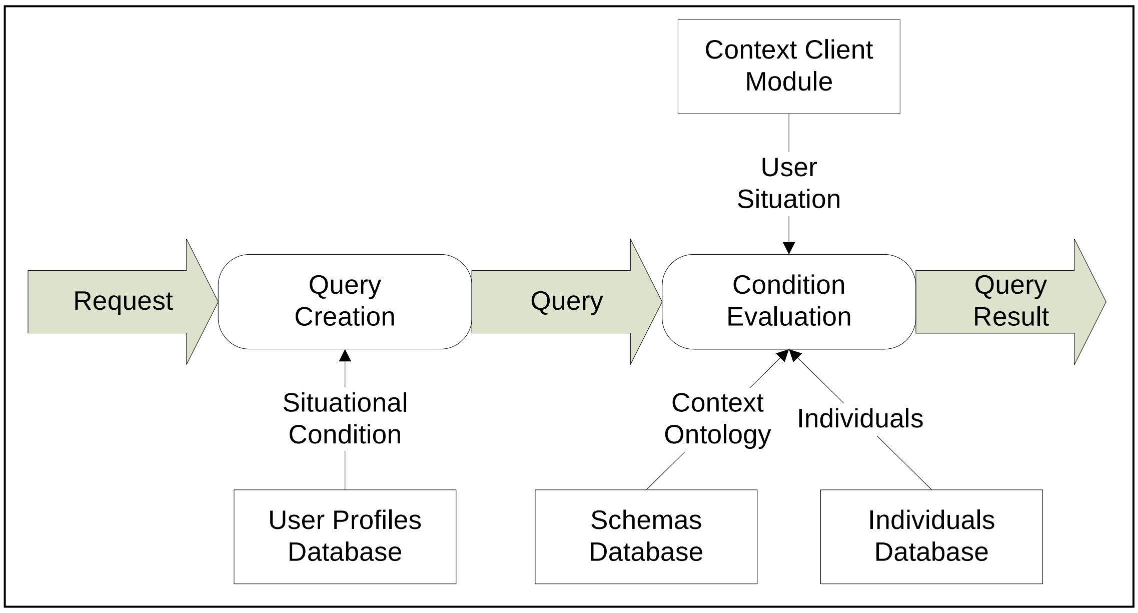


Figure 35: User Profile Selection Module

The actual query to be created depends on the situational condition, i.e. on the operator of the situational condition. For example, if we want to know whether a certain person is within a certain building, the query is different from that case, in which we want to know whether a certain person is in a certain type of room. The former case represents the location operator isWithin, the latter one represents the location operator isA. The forthcoming subsection 5.2 shows details of the created SPARQL queries and the corresponding ontology reasoning step. However, this means that the query creation process has to be carried out for each situational condition for all the Conditional Profile Subsets to be queried.

In the condition evaluation process, on the other hand, an ontology model is prepared that consists of the corresponding context ontology, e.g. the location ontology, the corresponding individuals and the actual user situation provided by the Context Client Module introduced in section 3.2.1. As mentioned in section 3.2.1, we assume that the received user situation is provided as a high-level context description, e.g. a certain room or a certain user activity, instead of low-level context descriptions such as GPS (Global Positioning System) information. Furthermore, we assume that the description of the provided user situation is provided as an RDF document, that it is based on the context ontology of section 4.3 and that the user situation is accurate.

Afterwards, this overall ontology model is queried by means of the corresponding SPARQL query, particularly a SPARQL ASK query. The specification of the SPARQL query language distinguishes between four different types of queries. The query type used in Figure 22 is the SELECT type, which returns all of the variables in a query. The ASK type, in comparison to this, which is used within the User Profile Selection Module, is used to test whether or not a query pattern has a solution. As a result, no information is returned about the possible query solutions, just whether or not a solution exists. The result contains either the Boolean value true or false. The other two query types are not used for our purposes, and hence are not explained. However, they are also described in the corresponding SPARQL specification [23].

The context ontology, i.e. the location ontology, user activity ontology or other ontologies for describing user context, that is loaded from the Schemas Database for the condition evaluation process in Figure 35 is the same for all users. However, the individuals such as rooms and buildings that are loaded from the Individuals Database are specific to the corresponding user. That is, we only load those individuals, e.g. rooms, buildings, etc. that are related to the corresponding user. Hence, each user has an own Individuals Database that models the user's world, i.e. the locations the user often visits and stays at, such as her home, the premises in which she works, and the environment in which she lives.

5.1.2 Selection Mechanism

Figure 36 shows the activity diagram, i.e. the algorithm, for the user profile selection mechanism. The activity diagram is explained in detail below.

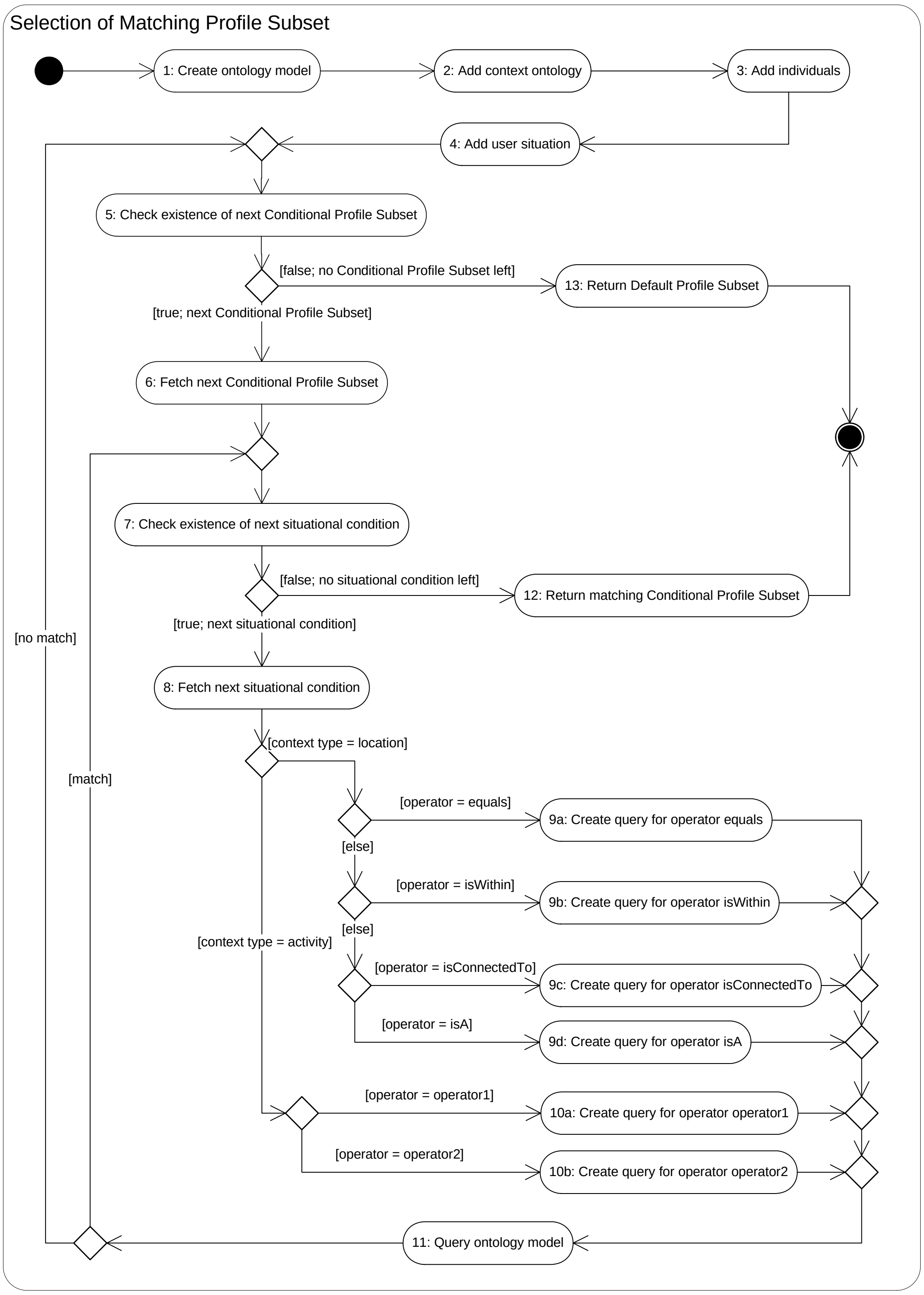

Selection of Matching Profile Subset
1: Create ontology model
2: Add context ontology
3: Add individuals
4: Add user situation
5: Check existence of next Conditional Profile Subset
[false; no Conditional Profile Subset left]
13: Return Default Profile Subset
[true; next Conditional Profile Subset]
6: Fetch next Conditional Profile Subset
7: Check existence of next situational condition
[false; no situational condition left]
12: Return matching Conditional Profile Subset
[true; next situational condition]
[no match]
8: Fetch next situational condition
[context type = location]
[match]
[operator = equals]
9a: Create query for operator equals
[else]
[operator = isWithin]
9b: Create query for operator isWithin
[else]
[context type = activity]
[operator = isConnectedTo]
9c: Create query for operator isConnectedTo
[operator = isA]
9d: Create query for operator isA
[operator = operator1]
10a: Create query for operator operator1
[operator = operator2]
10b: Create query for operator operator2
11: Query ontology model

Figure 36: Activity Diagram for User Profile Selection Mechanism

Before describing the single activities, some assumptions have to be made for the depicted algorithm. Firstly, we assume that there is only one matching Conditional Profile Subset for each user situation at any time. Hence, we assume that already at the creation time of Conditional Profile Subsets and the related situational conditions, conflicting and overlapping situational conditions are detected so that the creation is refused in these cases. Secondly, in entering the starting point of the activity diagram, we assume that there is at least one

Conditional Profile Subset to be queried. The starting point of this activity diagram represents the entering of activity 6 (Find matching Conditional Profile Subset) in the activity diagram of Figure 21 in section 3.2.1, in which we already know that there is at least one Conditional Profile Subset. Hence, this activity diagram is connected to the activity diagram of Figure 21 in providing the details of the activity 6 (Find matching Conditional Profile Subset) of Figure 21. Our third assumption is that a Conditional Profile Subset has at least one situational condition. In the following, the activities in Figure 36 are explained:

Activity 1 (Create ontology model): In this step, an initially empty ontology model is created. This is done with the Jena2 library [24] [25], particularly with the library version 2.5.4. All the subsequent ontology processing steps (except ontology reasoning steps) are also carried out with this software library.

Activity 2 (Add context ontology): In this step, the required context ontologies, i.e. the location ontology, user activity ontology or other ontologies for describing user context, is added to the ontology model of activity 1.

Activity 3 (Add individuals): In this step, the corresponding individuals from the user-specific Individuals Database, e.g. rooms, buildings, etc. are added to the resulting ontology model of activity 2.

Activity 4 (Add user situation): In this step, the current user's situation is added to the ontology model of activity 3. For this purpose, the current user's situation has been provided by the Context Client Module, see section 3.2.1 and activity 5 (Request user context of identified context types) of Figure 21. Now, the ontology model is complete.

Activity 5 (Check existence of next Conditional Profile Subset): In this step, the User Profile Selection Module checks the existence of remaining Conditional Profile Subsets. Based on one of the above assumptions, there is always at least one Conditional Profile Subset at the beginning. If a Conditional Profile Subset is still left, the algorithm continues with activity 6, otherwise with activity 13.

Activity 6 (Fetch next Conditional Profile Subset): In this step, the User Profile Selection Module fetches the next of the remaining Conditional Profile Subsets and continues with activity 7.

Activity 7 (Check existence of next situational condition): In this step, the User Profile Selection Module checks the existence of remaining situational conditions within the current Conditional Profile Subset object. Based on one of the above assumptions, there is always at least one situational condition for each Conditional Profile Subset. If a situational condition is left, the algorithm continues with activity 8, otherwise with activity 12.

Activity 8 (Fetch next situational condition): In this step, the User Profile Selection Module fetches the next of the remaining situational conditions of a Conditional Profile Subset and continues with activity 9a, b, c or d, or activity 10a or b, depending on the context type and location operator of the situational condition. In fact, there could be more decision points for distinguishing also between additional context types, which are not shown in Figure 36. The reason for distinguishing between different context types is that the query to be created in activity 9 or 10 is different for each context type.

Activity 9a-d (Create query for location operators): In these steps, the actual query, the SPARQL ASK query as introduced above, is created for the context type location. As the query depends on the context type of the situational condition, as well as the corresponding operator, there is a single activity for each different operator. In fact, there could be additional activities for additional operators for the context type location, which are not shown in Figure 36.

Activity 10a-b (Create query for activity operators): This step is comparable with activity 9a-d, but for the context type user activity.

Activity 11 (Query ontology model): Finally, the ontology model created in activity 4 is queried with the SPARQL ASK query of activity 9a-d and 10a-b respectively. This is the step, in which ontology reasoning takes place. For this purpose, an ontology reasoner is required. In our implementation, we use different ontology reasoning libraries, which are introduced in the evaluation chapter 6. If the query result is false, the algorithm continues with activity 5, in which it checks whether there is another Conditional Profile Subset. If the query result is true, the algorithm continues with activity 7. Now, the remaining situational conditions of the current Conditional Profile Subset have to be evaluated as well. As mentioned in section 3.1.1, our implementation interprets the existence of several situational conditions within a Conditional Profile Subset as conjunct to each other. Hence, all situational conditions of a Conditional Profile Subset have to be evaluated as true in order to identify a Conditional Profile Subset to be true.

Activity 12 (Return matching Conditional Profile Subset): In this step, the matching Conditional Profile Subset has been found and is returned. This activity is only entered in case all situational conditions of a Conditional Profile Subset have been evaluated as true. It is important here to refer to the above described assumptions that on the one hand a Conditional Profile Subset always has at least one situational condition. On the other hand, we can finish the prior evaluation loop as we know that there can not be another matching Conditional Profile Subset.

Activity 13 (Return Default Profile Subset): In this step, the Default Profile Subset is returned. This is because none of the evaluated Conditional Profile Subsets matches the user's current situation and because there is no further Conditional Profile Subset left for evaluation.

5.2 Ontology Reasoning Examples

In this section, we show details of the query process, i.e. details of activity 11 of the previous section 5.1. For each of the introduced example operators, we show an example condition, the corresponding SPARQL query [23], and the ontology reasoning steps, i.e. the inferred ontology statements. It should be mentioned that the ontology reasoning examples shown in the forthcoming subsections only cover a small subset of the ontology reasoning features supported by OWL.

All forthcoming SPARQL queries have several items in common. Firstly, all SPARQL queries are of the ASK query type, as already explained in the previous section 5.1. Secondly, SPARQL queries are encoded as triple patterns. Triple patterns are like RDF triples [17] except that each of the subject, predicate and object may be a variable. Thirdly, prefixes for namespaces are introduced at the beginning of the queries. The prefix *prof* represents the namespace for UPOS, i.e. the user profile ontology introduced in section 4.2. The prefix *context* represents the namespace of the location ontology introduced in section 4.3. The prefix *ind* represents the namespace for the individuals of the location ontology, and the prefix *rdf* represents the namespace for the RDF vocabulary [18]. Fourthly, the *entityID* and *contextValue* parameters delimited by quotation marks and plus signs in the depicted queries have to be substituted with the actual *entityID* and *contextValue* parameters of the corresponding situational condition during runtime. Fifthly, the parameters with a question mark at the beginning represent variables.

5.2.1 Equals Operator

Example situational condition: If the location of user Bob equals MeetingRoom_Comtec.

Example user situation: The location of user Bob is MeetingRoom_Comtec.

Obviously, the example user situation matches the situational condition. As explained in the previous section 5.1, the user situation is added to the overall ontology model that is then queried with a SPARQL query.

```
PREFIX prof: <http://ws.comtec.e-technik.uni-kassel.de/upos.owl#>
PREFIX ind: <http://ws.comtec.e-technik.uni-kassel.de/context-kb.owl#>
ASK
{ ind:”+entityID+” prof:hasLocation ind:”+contextValue+”
}
```

Figure 37: SPARQL Query for the equals Operator

The SPARQL query for the *equals* operator is shown in Figure 37. Substituting the *entityID* parameter with Bob and the *contextValue* parameter with MeetingRoom_Comtec, the ASK construct asks for the validity of the statement:

ind:Bob prof:hasLocation ind:MeetingRoom_Comtec

In this example, the searched statement is already available within the created ontology model, as the statement is exactly the user situation that has been added to the ontology model beforehand. Hence, in this example, no ontology reasoning has to take place, as the information is already available. Hence, the query result is true, which indicates a match between the example user situation and the situational condition.

Whereas this example is straightforward, the following variation is a bit more complex. Let us assume that there is a room called MeetingRoom_2413 in the ontology and that this room represents the same room as MeetingRoom_Comtec, which could be specified by means of the owl:sameAs statement. With this OWL [20] construct, we can refer to the same room with different identifiers, such as an official room name used in a floor plan of a building and an unofficially used room name like “our meeting room” that is related to its functionality.

Example situational condition: If the location of user Bob equals MeetingRoom_Comtec.

Example user situation: The location of user Bob is MeetingRoom_2413.

At a first glance, there is no match between this example user situation and the situational condition. However, the resulting match, i.e. the searched SPARQL ASK query, can be inferred from the existing statements 1 and 2 shown below, as ind:MeetingRoom_2413 can be substituted with ind:MeetingRoom_Comtec. Statement 1 is the user situation added to the queried ontology model beforehand, and statement 2 is part of the Individuals Database introduced in section 5.1:

1) ind:Bob prof:hasLocation ind:MeetingRoom_2413

2) ind:MeetingRoom_Comtec owl:sameAs ind:MeetingRoom_2413

=>

ind:Bob prof:hasLocation ind:MeetingRoom_Comtec (inferred from 1, 2)

5.2.2 IsWithin Operator

Example situational condition: If the location of user Bob isWithin Department_Comtec.

Example user situation: The location of user Bob is MeetingRoom_2413.

At a first glance, again, there is no match between this example user situation and the situational condition.

```
PREFIX prof: <http://ws.comtec.e-technik.uni-kassel.de/upos.owl#>
PREFIX context: <http://ws.comtec.e-technik.uni-kassel.de/context.owl#>
PREFIX ind: <http://ws.comtec.e-technik.uni-kassel.de/context-kb.owl#>
ASK
{ { ind:"+entityID+" prof:hasLocation ind:"+contextValue+" }
UNION
{ ind:"+entityID+" prof:hasLocation ?userLocation .
  ?userLocation context:isPartOf ind:"+contextValue+" }
}
```

Figure 38: SPARQL Query for the isWithin Operator

Figure 38 depicts the SPARQL query for the *isWithin* operator. The ASK construct consists of a UNION construct that represents the disjunction of two queries. The first query is the same as for the *equals* operator in order to check equality. The second one checks whether a location is part of another location. In fact, the first query is only needed because the current OWL standard [20] does not support reflexive relationships as would be needed here for the context:isPartOf relationship. Substituting the *entityID* parameter of Figure 38 with Bob and the *contextValue* parameter with Department_Comtec, the second query for this example is as follows:

ind:Bob prof:hasLocation ?userLocation

?userLocation context:isPartOf ind:Department_Comtec

This second query has a solution starting with the below initial ontology statements 1 to 5. As the two queries are connected as a disjunction, the whole query has a solution. Statement 1 is the user situation added to the queried ontology model beforehand. Statements 2 to 3 are part of the Individuals Database, and statements 4 and 5 are part of the specification of the location ontology as defined in section 4.3. The finally inferred statements can be inferred by means of different inference steps.

In the first step, statements 6 to 7 can be inferred by applying the rdfs:subPropertyOf relationship of statement 4. In the next step, the final statements can be inferred by applying the transitivity of the context:isPartOf relationship of statement 5 to statements 6 and 7.

1) ind:Bob prof:hasLocation ind:MeetingRoom_2413

2) ind:MeetingRoom_2413 context:isDirectPartOf ind:Floor_A-3

3) ind:Floor_A-3 context:isDirectPartOf ind:Department_ComTec

4) context:isDirectPartOf rdfs:subPropertyOf context:isPartOf

5) context:isPartOf rdf:type owl:TransitiveProperty

=>

6) ind:MeetingRoom_2413 context:isPartOf ind:Floor_A-3 (inferred from 2, 4)

7) ind:Floor_A-3 context:isPartOf ind:Department_ComTec (inferred from 3, 4)

=>

ind:Bob prof:hasLocation ind:MeetingRoom_2413 (same as 1)

ind:MeetingRoom_2413 context:isPartOf ind:Department_Comtec (inferred from 5, 6, 7)

5.2.3 IsConnectedTo Operator

Example situational condition: If the location of user Bob isConnectedTo PrinterSpace_2419.

Example user situation: The location of user Bob is TwoPersonOffice_2414.

At a first glance, again, there is no match between this example user situation and the situational condition.

```
PREFIX prof: <http://ws.comtec.e-technik.uni-kassel.de/upos.owl#>
PREFIX context: <http://ws.comtec.e-technik.uni-kassel.de/context.owl#>
PREFIX ind: <http://ws.comtec.e-technik.uni-kassel.de/context-kb.owl#>
ASK
{ { ind:"+entityID+" prof:hasLocation ind:"+contextValue+" }
UNION
{ ind:"+entityID+" prof:hasLocation ?userLocation .
  ?userLocation context:isConnectedTo ind:"+contextValue+" }
}
```

Figure 39: SPARQL Query for the isConnectedTo Operator

Figure 39 depicts the SPARQL query for the isConnectedTo operator. The ASK construct consists of a UNION construct that represents the disjunction of two queries. The first query is the same as for the *equals* operator in order to check equality. The second one checks whether a room is connected to another room. In fact, the first query is again only needed because the current OWL standard [20] does not support reflexive relationships as would be needed here for the context:isConnectedTo relationship. Substituting the *entityID* parameter of Figure 39 with Bob and the *contextValue* parameter with PrinterSpace_2419, the second query for this example is as follows:

ind:Bob prof:hasLocation ?userLocation

?userLocation context:isConnectedTo ind:PrinterSpace_2419

This second query has a solution starting with the below initial ontology statements 1 to 8. As the two queries are connected as a disjunction, the whole query has a solution. Statement 1 is the user situation added to the queried ontology model beforehand. Statements 2 to 5 are part of the Individuals Database, and statements 6 to 8 are part of the specification of the location ontology as defined in section 4.3. Again, the finally inferred statements can be inferred by means of different inference steps.

In the first step, statements 9, 11 and 12 can be inferred by applying the rdfs:subPropertyOf relationship of statement 6. Statement 10, on the other hand, can be inferred by applying the symmetry of the context:isDirectlyConnectedTo relationship of statement 7. In the next step, statement 13 can be inferred by again applying the rdfs:subPropertyOf relationship of statement 6, and statement 14 can be inferred by applying the transitivity of the context:isConnectedTo relationship of statement 8. In the next steps, statement 15 and the final statements can be inferred by again applying the transitivity of the context:isConnectedTo relationship.

1) ind:Bob prof:hasLocation ind:TwoPersonOffice_2414

2) ind:TwoPersonOffice_2414 context:isDirectlyConnectedTo ind:Corridor_Comtec

3) ind:MeetingRoom_2413 context:isDirectlyConnectedTo ind:Corridor_Comtec

4) ind:MeetingRoom_2413 context:isDirectlyConnectedTo ind:Kitchen_2415

5) ind:Kitchen_2415 context:isDirectlyConnectedTo ind:PrinterSpace_2419

6) context:isDirectlyConnectedTo rdfs:subPropertyOf context:isConnectedTo

7) context:isDirectlyConnectedTo rdf:type owl:SymmetricProperty

8) context:isConnectedTo rdf:type owl:TransitiveProperty

=>

9) ind:TwoPersonOffice_2414 context:isConnectedTo ind:Corridor_Comtec (inferred from 2, 6)

10) ind:Corridor_Comtec context:isDirectlyConnectedTo ind:MeetingRoom_2413 (inferred from 3, 7)

11) ind:MeetingRoom_2413 context:isConnectedTo ind:Kitchen_2415 (inferred from 4, 6)

12) ind:Kitchen_2415 context:isConnectedTo ind:PrinterSpace_2419 (inferred from 5 and 6)

=>

13) ind:Corridor_Comtec context:isConnectedTo ind:MeetingRoom_2413 (inferred from 6, 10)

14) ind:MeetingRoom_2413 context:isConnectedTo ind:PrinterSpace_2419 (inferred from 8, 11, 12)

=>

15) ind:TwoPersonOffice_2414 context:isConnectedTo ind:MeetingRoom_2413 (inferred from 8, 9, 13)

=>

ind:Bob prof:hasLocation ind:TwoPersonOffice_2414 (same as 1)

ind:TwoPersonOffice_2414 context:isConnectedTo ind:PrinterSpace_2419 (inferred from 8, 14, 15)

5.2.4 IsA Operator

Example situational condition: If the location of user Bob isA Workspace_Business.

Example user situation: The location of user Bob is SinglePersonOffice_30.

At a first glance, again, there is no match between this example user situation and the situational condition.

```
PREFIX prof: <http://ws.comtec.e-technik.uni-kassel.de/upos.owl#>
PREFIX context: <http://ws.comtec.e-technik.uni-kassel.de/context.owl#>
PREFIX ind: <http://ws.comtec.e-technik.uni-kassel.de/context-kb.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
ASK
{ ind:"+entityID+" prof:hasLocation ?userLocation .
  ?userLocation rdf:type context:"+contextValue+"
}
```

Figure 40: SPARQL Query for the isA Operator

Figure 40 depicts the SPARQL query for the isA operator. Substituting the *entityID* parameter with Bob and the *contextValue* parameter with Workspace_Business, the ASK construct for this example is as follows:

ind:Bob prof:hasLocation ?userLocation

?userLocation rdf:type context:Workspace_Business

This query has a solution starting with the below initial ontology statements 1 to 4. Statement 1 is the user situation added to the queried ontology model beforehand. Statement 2 is part of the Individuals Database, and statements 3 to 4 are part of the specification of the location ontology as defined in section 4.3. Again, the finally inferred statements can be inferred by means of different inference steps.

In the first step, statement 5 can be inferred by applying the rdfs:subClassOf relationship of statement 3. In the final step, the generalisation / specialisation relationship rdfs:subClassOf can be applied again.

1) ind:Bob prof:hasLocation ind:SinglePersonOffice_30

2) ind:SinglePersonOffice_30 rdf:type context:SinglePersonOffice

3) context:SinglePersonOffice rdfs:subClassOf context:Office

4) context:Office rdfs:subClassOf context:Workspace_Business

=>

5) ind:SinglePersonOffice_30 rdf:type context:Office (inferred from 2, 3)

=>

ind:Bob prof:hasLocation ind:SinglePersonOffice_30 (same as 1)

ind:SinglePersonOffice_30 rdf:type context:Workspace_Business (inferred from 4, 5)

In the following, we also show a second ontology reasoning example. In this example, the owl:equivalentClass construct is included. This construct is similar to the owl:sameAs construct that was introduced above. However, whereas the owl:sameAs construct can be used for defining two individuals as equal, the owl:equivalentClass can instead be used to define two ontology concepts as equivalent. In the location ontology introduced in section 4.3, e.g. the concepts PresentationSpace and VideoProjectorSpace have been defined equivalent as both terms are usually used to describe the same kind of room.

Example situational condition: If the location of user Bob isA VideoProjectorSpace.

Example user situation: The location of Bob is MeetingRoom_Comtec.

At a first glance, there is again no match between this example user situation and the situational condition. However, the corresponding query has a solution starting with the below initial ontology statements 1 to 4. Statement 1 is the user situation added to the queried ontology model beforehand. Statements 2 and 3 are part of the Individuals Database, and statement 4 is part of the specification of the location ontology as defined in section 4.3. Statement 5 can be inferred from statement 2 and 3 as MeetingRoom_2413 can be substituted with MeetingRoom_Comtec. In the final step, the missing statement to satisfy the corresponding SPARQL query can be inferred from statement 4 and 5 as PresentationSpace can be substituted with VideoProjectorSpace.

1) ind:Bob prof:hasLocation ind:MeetingRoom_Comtec

2) ind:MeetingRoom_Comtec owl:sameAs ind:MeetingRoom_2413

3) ind:MeetingRoom_2413 rdf:type context:PresentationSpace

4) context:PresentationSpace owl:equivalentClass context:VideoProjectorSpace

=>

5) ind:MeetingRoom_Comtec rdf:type context:PresentationSpace (inferred from 2, 3)

=>

ind:Bob prof:hasLocation ind:MeetingRoom_Comtec (same as 1)

ind:MeetingRoom_Comtec rdf:type context:VideoProjectorSpace (inferred from 4, 5)

5.2.5 IsWithinA Operator

Example situational condition: If the location of user Bob isWithinA WlanAccessRange_Business.

Example user situation: The location of user Bob is MeetingRoom_Comtec.

At a first glance, again, there is no match between this example user situation and the situational condition.

```
PREFIX prof: <http://ws.comtec.e-technik.uni-kassel.de/upos.owl#>
PREFIX context: <http://ws.comtec.e-technik.uni-kassel.de/context.owl#>
PREFIX ind: <http://ws.comtec.e-technik.uni-kassel.de/context-kb.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
ASK
{ { ind:"+entityID+" prof:hasLocation ?userLocation .
    ?userLocation rdf:type context:"+contextValue+" }
UNION
{ ind:"+entityID+" prof:hasLocation ?userLocation .
  ?userLocation context:isPartOf ?location2 .
  ?location2 rdf:type context:"+contextValue+" }
}
```

Figure 41: SPARQL Query for the isWithinA Operator

Figure 41 depicts the SPARQL query for the isWithinA operator. The ASK construct consists of a UNION construct that represents the disjunction of two queries. The first query is the same as for the *isA* operator. The second one checks whether a location is part of another location of a certain location type. In fact, the first query is again only needed because the current OWL standard [20] does not support reflexive relationships as would be needed here for the context:isPartOf relationship. Substituting the *entityID* parameter of Figure 41 with Bob and the *contextValue* parameter with WlanAccessRange_Business, the second query for this example is as follows, consisting of two variables this time:

ind:Bob prof:hasLocation ?userLocation

?userLocation context:isPartOf ?location2

?location2 rdf:type context:WlanAccessRange_Business

This second query has a solution starting with the below initial ontology statements 1 to 7. As the two queries are connected as a disjunction, the whole query has a solution. Statement 1 is the user situation added to the queried ontology model beforehand. Statements 2 to 5 are part of the Individuals Database, and statements 6 and 7 are part of the specification of the location ontology as defined in section 4.3. The finally inferred statements can be inferred by means of different inference steps.

In the first step, statements 8 and 9 can be inferred by applying the rdfs:subPropertyOf relationship of statement 6. In the next step, statement 9 can be inferred by applying the transitivity of the context:isPartOf relation of statement 7. In the final step, the owl:sameAs relationship of statement 2 can be applied to statement 10.

1) ind:Bob prof:hasLocation ind:MeetingRoom_Comtec

2) ind:MeetingRoom_Comtec owl:sameAs ind:MeetingRoom_2413

3) ind:MeetingRoom_2413 context:isDirectPartOf ind:Floor_A-3

4) ind:Floor_A-3 context:isDirectPartOf ind:WlanAccessRange_Comtec

5) ind:WlanAccessRange_Comtec rdf:type context:WlanAccessRange_Business

6) context:isDirectPartOf rdfs:subPropertyOf context:isPartOf

7) context:isPartOf rdf:type owl:TransitiveProperty

=>

8) ind:MeetingRoom_2413 context:isPartOf ind:Floor_A-3 (inferred from 3, 6)

9) ind:Floor_A-3 context:isPartOf ind:WlanAccessRange_Comtec (inferred from 4, 6)

=>

10) ind:MeetingRoom_2413 context:isPartOf ind:WlanAccessRange_Comtec
(inferred from 7, 8, 9)

=>

ind:Bob prof:hasLocation ind:MeetingRoom_Comtec (same as 1) ind:MeetingRoom_Comtec context:isPartOf ind:WlanAccessRange_Comtec (inferred from 2, 10)

ind:WlanAccessRange_Comtec rdf:type context:WlanAccessRange_Business (same as 5)

5.2.6 IsConnectedToA Operator

Example situational condition: If the location of user Bob isConnectedToA PrinterSpace_Business.

Example user situation: The location of Bob is TwoPersonOffice_2414.

At a first glance, again, there is no match between this example user situation and the situational condition.

```
PREFIX prof: <http://ws.comtec.e-technik.uni-kassel.de/upos.owl#>
PREFIX context: <http://ws.comtec.e-technik.uni-kassel.de/context.owl#>
PREFIX ind: <http://ws.comtec.e-technik.uni-kassel.de/context-kb.owl#>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
ASK
{ { ind:"+entityID+" prof:hasLocation ?userLocation .
    ?userLocation rdf:type context:"+contextValue+" }
UNION
{ ind:"+entityID+" prof:hasLocation ?userLocation .
  ?userLocation context:isConnectedTo ?location2 .
  ?location2 rdf:type context:"+contextValue+" }
}
```

Figure 42: SPARQL Query for the isConnectedToA Operator

Figure 42 depicts the SPARQL query for the isConnectedToA operator. The ASK construct consists of a UNION construct that represents the disjunction of two queries. The first query is the same as for the *isA* operator. The second one checks whether a room is connected to another room of a certain type. In fact, the first query is again only needed because the current OWL standard [20] does not support reflexive relationships as would be needed here for the context:isConnectedTo relationship. Substituting the *entityID* parameter of Figure 42 with Bob and the *contextValue* parameter with PrinterSpace_Business, the second query for this example is as follows, again consisting of two variables this time:

ind:Bob prof:hasLocation ?userLocation

?userLocation context:isConnectedTo ?location2

?location2 rdf:type context:PrinterSpace_Business

This second query has a solution starting with the below initial ontology statements 1 to 9. As the two queries are connected as a disjunction, the whole query has a solution. Statement 1 is the user situation added to the queried ontology model beforehand. Statements 2 to 6 are part of the Individuals Database, and statements 7 to 9 are part of the specification of the location ontology as defined in section 4.3. Again, the finally inferred statements can be inferred by means of different inference steps.

In the first step, statements 10, 12 and 13 can be inferred by applying the rdfs:subPropertyOf relationship of statement 7. Statement 11, on the other hand, can be inferred, by applying the symmetry of the context:isDirectlyConnectedTo relationship of statement 8. In the next step, statement 14 can be inferred by again applying the rdfs:subPropertyOf relationship of statement 7, and statement 15 can be inferred by applying the transitivity of the context:isConnectedTo relationship of statement 9. In the next steps, statement 16 and the final statements can be inferred by again applying the transitivity of the context:isConnectedTo relationship.

1) ind:Bob prof:hasLocation ind:TwoPersonOffice_2414

2) ind:TwoPersonOffice_2414 context:isDirectlyConnectedTo ind:Corridor_Comtec

3) ind:MeetingRoom_2413 context:isDirectlyConnectedTo ind:Corridor_Comtec

4) ind:MeetingRoom_2413 context:isDirectlyConnectedTo ind:Kitchen_2415

5) ind:Kitchen_2415 context:isDirectlyConnectedTo ind:PrinterSpace_2419

6) ind:PrinterSpace_2419 rdf:type context:PrinterSpace_Business

7) context:isDirectlyConnectedTo rdfs:subPropertyOf context:isConnectedTo

8) context:isDirectlyConnectedTo rdf:type owl:SymmetricProperty

9) context:isConnectedTo rdf:type owl:TransitiveProperty

=>

10) ind:TwoPersonOffice_2414 context:isConnectedTo ind:Corridor_Comtec (inferred from 2, 7)

11) ind:Corridor_Comtec context:isDirectlyConnectedTo ind:MeetingRoom_2413 (inferred from 3, 8)

12) ind:MeetingRoom_2413 context:isConnectedTo ind:Kitchen_2415 (inferred from 4, 7)

13) ind:Kitchen_2415 context:isConnectedTo ind:PrinterSpace_2419 (inferred from 5 and 7)

=>

14) ind:Corridor_Comtec context:isConnectedTo ind:MeetingRoom_2413 (inferred from 7, 11)

15) ind:MeetingRoom_2413 context:isConnectedTo ind:PrinterSpace_2419 (inferred from 9, 12, 13)

=>

16) ind:TwoPersonOffice_2414 context:isConnectedTo ind:MeetingRoom_2413 (inferred from 9, 10, 14)

=>

ind:Bob prof:hasLocation ind:TwoPersonOffice_2414 (same as 1)

ind:TwoPersonOffice_2414 context:isConnectedTo ind:PrinterSpace_2419 (inferred from 9, 15, 16)

ind:PrinterSpace_2419 rdf:type context:PrinterSpace_Business (same as 6)

5.3 Summary

In the first part of this chapter, we have introduced our User Profile Selection Module, which has already been mentioned in section 3.2.1 as part of our user profile management framework. The User Profile Selection Module carries out the selection of the matching user profile subset for the user's current situation. This is done by applying ontology reasoning in the selection process. First, we have explained the general functioning of the module by means of its sub-modules and processing steps. Afterwards, we have depicted the actual selection algorithm by means of a detailed activity diagram.

In the second part of this chapter, we have depicted the details of the query process. That is, for each example operator introduced in section 4.3.1, we have shown one or two detailed query processes. In particular, we have created example user situations and example situational conditions as a starting point for the query process. Afterwards, the corresponding SPARQL query has been introduced. Finally, possible solutions for the inference step have been shown that satisfy the corresponding query. This has been done by explaining the inferred ontology statements in a step by step process from the starting point until the final success state.

6 Evaluation of the User Profile Selection Mechanism

In this chapter, we evaluate our user profile selection approach in terms of runtime performance. For this purpose, the execution time is compared between different reasoning libraries. Besides different reasoning libraries, we also compare the execution time concerning other variables related to ontology processing such as the number of individuals in the Individuals Database and different interrelations between individuals in the ontology. These comparisons are carried out for various reasoning examples, which are also introduced in this chapter. Furthermore, we compare the supported functionality of our user profile selection approach with other approaches that are not based on ontology reasoning. In the first part, we describe the setup of the measurements. In the second section, we then show the results of the measurements.

6.1 Setup

This section describes the setup for our measurements. This includes the system environment of the user profile management system, the characteristics of our Individuals Database as well as the description of the measurement steps and measurement samples.

6.1.1 System Environment

All measurements have been carried out on the same system, i.e. the same hardware and software environment. This system has the following characteristics:

Computer Type:
- IBM ThinkPad T43, Type 1871 – 4AG

Operating System:
- Microsoft Windows XP Professional, Version 2002, Service Pack 2

Central Processing Unit:
- Intel Pentium M, 1.86 GHz

Main Memory:
- 1 GB RAM

The implementation of our whole profile management system has been done with the Java programming language from Sun Microsystems, version Standard Edition 1.5.0. The maximum amount of memory that the Java virtual machine has attempted to use in the default configuration has been 63.5625 MBytes, as the Runtime.getRuntime().maxMemory() method call has revealed.

In addition, we have used the Java-based Jena2 library [24] [25], particularly version 2.5.4., for ontology processing steps such as read-in, parsing and querying. For the ontology reasoning task, we have also used the Jena2 library, but also other libraries, which are introduced in the forthcoming section 6.2. Finally, we have also used the Java-based log4j library from the Apache Logging Services Project [89] [90], particularly version 1.2.12. This library is also part of the Jena2 project and hence does not have to be imported separately anymore. The log4j library has been used to log and, hence, to measure the execution time for different execution steps.

There are two things, which should be mentioned with regard to logging with log4j. Firstly, logging events require execution time themselves. However, the execution time for logging events is an issue of few microseconds, as experimental results with the log4j version 1.1.3 on a system with Windows 2000 operating system, an AMD Duron central processing unit clocked at 800 MHz, and Java version Standard Edition 1.3 from Sun Microsystems have shown. This issue is documented in the Java documentation (JavaDoc) of log4j version 1.1.3 and on the Web page for the log4j version 1.2 of the Apache Logging Services Project [89]. As our measurement results, which are shown in the forthcoming section 5.2, are within three-digit milliseconds and few seconds, few logging events do not influence our measurement results.

Secondly, log4j-based logging of execution time returns the difference, measured in milliseconds, between the current time and a certain point in time in the past, in particular 1970. However, while the unit of time of the return value is a millisecond, the granularity of the value depends on the underlying operating system and may be larger. For example, some operating systems measure time in units of tens of milliseconds. Some others measure time in units of approximately 15 to 16 milliseconds. This is explained in detail in [91]. Our system belongs to the latter group and measures time in units of approximately 15 to 16 milliseconds. Consequently, all measurements shown in the subsequent section may have a deviation of approximately 15 to 16 milliseconds. However, as mentioned above, our measurement results are within three-digit milliseconds and few seconds. Hence, they are only slightly influenced by this deviation.

6.1.2 Individuals Database

In section 4.2 and 4.3, we have introduced our user profile ontology and our location ontology respectively. The corresponding specifications are provided in Appendix C: Specification of the User Profile Ontology and Appendix E: Specification of the Location Ontology respectively. However, these ontologies define schemas, but do not include any individuals. The reason for this was that these two ontologies are independent of a particular user, whereas the individuals of the ontology concepts are user-specific.

In section 5.1, we have also mentioned the user-specific Individuals Database that is loaded by the user profile selection mechanism. For the forthcoming measurements, we have created an Individuals Database that includes a model of our premises, i.e. rooms, corridors, floors, wings, departments, buildings, WLAN access ranges, UMTS access ranges, and other locations. We have not only created individual locations, but also defined relationships between connected rooms and locations that include other locations. As described in section 4.3, our location ontology can be used for such definitions.

Table 5: Characteristics of the Ontologies

Ontology	*# Triples*	*# Classes*	*# Properties*	*# Individuals*
User Profile Ontology	135	11	15	0
Location Ontology	606	200	9	0
Individuals Database	749	88	3	247
Composed Ontology	1490	207	23	247

Table 5 shows the characteristics of the single ontologies and the combined ontology that comprises all three single ontologies. The combined ontology is the ontology that exists after carrying out activity 3 of the activity diagram in Figure 36. Hence, this is the ontology that is used during the user profile selection process, i.e. the ontology that is queried.

The number of triples of the ontologies has been counted by the W3C RDF Validation Service [94]. The number of classes, properties and individuals of the ontologies has been checked by the Pellet library [30]. In addition, the Pellet library has also been used for successfully checking the consistency of the ontologies. Whereas the number of triples for the composed ontology in Table 5 is the sum of those ones for the single ontologies, this is not the case for the number of classes and properties. The reason for this is that e.g. the 88 classes shown in the row of the Individuals Database are not defined in the Individuals Database but imported from the location ontology. Hence, the number of classes and properties depicted in the row of the composed ontology is smaller than the sum of the other rows.

For the subsequent measurements, we use the composed ontology as characterised here. However, we also carry out measurements with bigger Individuals Databases, which then lead to bigger composed ontologies. This is explained in more detail in the result section 6.2.

6.1.3 Measurement Steps

We have carried out different measurements concerning the user profile selection mechanism and the corresponding activity diagram introduced in section 5.1. There are basically two measurement steps.

Step 1

Step 1 includes the activities 1 to 3 of the activity diagram in Figure 36. Hence, this step represents the creation of the combined ontology model. In this step, first an empty Jena2 [24] [25] DefaultModel is created with the method call ModelFactory.createDefaultModel(). Afterwards, the user profile ontology introduced in section 4.2, the context ontology introduced in section 4.3 and the user-specific Individuals Database introduced in section 5.1 are separately read from the file system, i.e. from .owl files containing RDF/XML serialisations [82] of the related ontologies, and separately added to the combined ontology model. This is done with the method call model.read(inputStream, ""), in which the parameter inputStream represents the input stream (Java InputStream) of an ontology file. For more detailed documentation related to Jena2, please refer to the Jena2 Web page [92].

Whereas the user profile ontology and the context ontology can be applied for all users of the service platform, the Individuals Database is specific for each user of the service platform, as described in section 5.1. Hence, the resulting combined ontology model is also user-specific. This ontology model could be loaded once and kept in memory or stored as a whole as long as there are no changes either in the user profile ontology, the context ontology or the Individuals Database. This means, that this measurement step is actually not very interesting, as the required execution time does not have to be considered for each user profile selection process, but only for the first one.

Step 2

Step 2 includes the activities 4 and 8 to 11 of the activity diagram in Figure 36. The execution time of the activities 5 to 7 is insignificant in comparison to the execution time of the other activities in Figure 36, as they basically only comprise some few if-then-else statements. Furthermore, the execution time for these if-then-else statements is the same for all measurements and, hence, do not influence the forthcoming comparisons of the measured execution times.

The activities 8 to 11 represent the query creation and the actual query. In the query creation process, the query is created based on the given situational condition, i.e. based on the context type, the operator and the context value of the situational condition. Queries are expressed with the SPARQL query language [23]. The creation of such example SPARQL queries has been shown in section 5.2. Subsequently, the actual query is carried out. For this execution steps, different SPARQL query engines are used, i.e. different ontology reasoners are used that support SPARQL queries. The query process also includes some query preparation or query configuration process beforehand, which is specific to the corresponding ontology reasoners.

In order to ensure comparable measurement results for different reasoners, step 2 includes another processing sub-step, particularly the first processing sub-step in step 2. In this sub-step we assume that the combined ontology model created in step 1 above is buffered as a String object containing the RDF/XML serialisation of the combined ontology model. First, this String object is converted to the required internal model representation of the corresponding ontology reasoner, which is different for different reasoners. Afterwards, the activity 4, i.e. adding the current user situation to that model, and the activities 8 to 11, i.e. query creation and query, are carried out. Details of the ontology reasoner specific ontology model preparation are described in the subsequent section on ontology reasoners.

Each measurement depicted in the subsequent sections has been carried out 10 times. The shown measurement results, i.e. the execution times, are the arithmetic mean $\bar{x}$ of all 10 independent runs, computed with the following equation:

$$\bar{x} = \frac{x_1 + x_2 + \ldots + x_n}{n} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

In order to give additional qualitative information on the distribution of the corresponding measurement values, we also provide the corresponding range *R* between the maximum and the minimum value, and the standard deviation *s*, which are computed with the following equations [93]:

$$R = x_{\max} - x_{\min}$$

$$s = \sqrt{\frac{1}{n-1}\sum_{i=1}^{n}(x_i - \bar{x})^2}$$

6.1.4 Measurement Samples

In the previous subsection, we have described different measurement steps. However, we have not explained, which samples have been measured, instead this is done now. For each operator introduced in section 4.3.1, we have selected three samples, whereof the third sample is always a sample with no match, i.e. which represents the false case. Each sample gets a short identifier, e.g. A1 and B2, which is used later on in the document to refer to the samples. Some samples have already been introduced in section 5.2. If not already introduced, the corresponding user situation and situational condition are shown below. For the satisfiable samples, in addition, also the starting statements and one possible solution is shown that satisfies the corresponding query introduced in section 5.2, by means of the finally inferred statements. However, the intermediate steps of the inference process are not depicted. Details of intermediate steps are also depicted in section 5.2.

Equals operator

A1: Identical to the first example in section 5.2.1.

A2: Identical to the second example in section 5.2.1.

A3: As follows:

Situational condition: If the location of user Bob equals TwoPersonOffice_2414

User situation: The location of user Bob is MeetingRoom_2413

<u>IsWithin operator</u>

B1: Identical to the example in section 5.2.2.

B2: As follows:

Situational condition: If the location of user Bob isWithin UmtsAccessRange_T-Mobile

User situation: The location of user Bob is MeetingRoom_Comtec

Starting statements:

1) ind:Bob prof:hasLocation ind:MeetingRoom_Comtec

2) ind:MeetingRoom_Comtec owl:sameAs ind:MeetingRoom_2413

3) ind:MeetingRoom_2413 context:isDirectPartOf ind:Floor_A-3

4) ind:Floor_A-3 context:isDirectPartOf ind:Department_ComTec

5) ind:Department_ComTec context:isDirectPartOf ind:Premises_Eecs

6) ind:Premises_Eecs context:isDirectPartOf ind:Premises_University

7) ind:Premises_University context:isDirectPartOf ind:Kassel

8) ind:Kassel context:isDirectPartOf ind:UmtsAccessRange_T-Mobile

9) context:isDirectPartOf rdfs:subPropertyOf context:isPartOf

10) context:isPartOf rdf:type owl:TransitiveProperty

Finally inferred statements:

ind:Bob prof:hasLocation ind:MeetingRoom_Comtec

ind:MeetingRoom_Comtec context:isPartOf ind:UmtsAccessRange_T-Mobile

B3: As follows:

Situational condition: If the location of user Bob isWithin OfficeBuilding_WA66

User situation: The location of user Bob is MeetingRoom_Comtec

<u>IsConnectedTo operator</u>

C1: Identical to the example in section 5.2.3.

C2: As follows:

Situational condition: If the location of user Bob isConnectedTo PrinterSpace_2419

User situation: The location of user Bob is MeetingRoom_Comtec

Starting statements:

1) ind:Bob prof:hasLocation ind:MeetingRoom_Comtec

2) ind:MeetingRoom_Comtec owl:sameAs ind:MeetingRoom_2413

3) ind:MeetingRoom_2413 context:isDirectlyConnectedTo ind:Kitchen_2415

4) ind:Kitchen_2415 context:isDirectlyConnectedTo ind:PrinterSpace_2419

5) context:isDirectlyConnectedTo rdfs:subPropertyOf context:isConnectedTo

6) context:isConnectedTo rdf:type owl:TransitiveProperty

Finally inferred statements:

ind:Bob prof:hasLocation ind:MeetingRoom_Comtec

ind:MeetingRoom_Comtec context:isConnectedTo ind:PrinterSpace_2419

C3: As follows:

Situational condition: If the location of user Bob isConnectedTo Kitchen_28

User situation: The location of user Bob is MeetingRoom_Comtec

<u>IsA operator</u>

D1: Identical to the first example in section 5.2.4.

D2: Identical to the second example in section 5.2.4.

D3: As follows:

Situational condition: If the location of user Bob isA PrivatePlace

User situation: The location of user Bob is MeetingRoom_Comtec

<u>IsWithinA operator</u>

E1: As follows:

Situational condition: If the location of user Bob isWithinA OfficeBuilding

User situation: The location of user Bob is SinglePersonOffice_30

Starting statements:

1) ind:Bob prof:hasLocation ind:SinglePersonOffice_30

2) ind:SinglePersonOffice_30 context:isDirectPartOf ind:Floor_WA66-A-1

3) ind:Floor_WA66-A-1 context:isDirectPartOf ind:Wing_WA66-A

4) ind:Wing_WA66-A context:isDirectPartOf ind:OfficeBuilding_WA66

5) ind:OfficeBuilding_WA66 rdf:type context:OfficeBuilding

6) context:isDirectPartOf rdfs:subPropertyOf context:isPartOf

7) context:isPartOf rdf:type owl:TransitiveProperty

Finally inferred statements:

ind:Bob prof:hasLocation ind:SinglePersonOffice_30

ind:SinglePersonOffice_30 context:isPartOf ind:OfficeBuilding_WA66

ind:OfficeBuilding_WA66 rdf:type context:OfficeBuilding

E2: Identical to the example in section 5.2.5.

E3: As follows:

Situational condition: If the location of user Bob isWithinA BtAccessRange_Business

User situation: The location of user Bob is SinglePersonOffice_30

<u>IsConnectedToA operator</u>

F1: Identical to the example in section 5.2.6.

F2: As follows:

Situational condition: If the location of user Bob isConnectedToA Library_Business

User situation: The location of user Bob is SinglePersonOffice_2408

Starting statements:

1) ind:Bob prof:hasLocation ind:SinglePersonOffice_2408

2) ind:SinglePersonOffice_2408 context:isDirectlyConnectedTo ind:Corridor_Comtec

3) ind: Corridor_Comtec context:isDirectlyConnectedTo ind:MeetingRoom_2413

4) ind:Library_2413 owl:sameAs ind:MeetingRoom_2413

5) ind:Library_2413 rdf:type context:Library_Business

6) context:isDirectlyConnectedTo rdfs:subPropertyOf context:isConnectedTo

7) context:isConnectedTo rdf:type owl:TransitiveProperty

Finally inferred statements:

ind:Bob prof:hasLocation ind:SinglePersonOffice_2408

ind:SinglePersonOffice_2408 context:isConnectedTo ind:Library_2413

ind:Library_2413 rdf:type context:Library_Business

F3: As follows:

Situational condition: If the location of user Bob isConnectedToA PrinterSpace_Private

User situation: The location of user Bob is SinglePersonOffice_2408

6.2 Results

In this section, we provide the results of our user profile selection approach in terms of runtime performance. Our results are shown for different ontology reasoning libraries, different reasoning samples, different numbers of individuals in the Individuals Database, different interrelations between these individuals, and different user profile selection approaches. Furthermore, we compare the supported functionality with other user profile selection approaches that are not based on ontology reasoning.

In section 6.1.3, we have introduced two measurement steps. The first measurement step, i.e. the creation of the combined ontology model, is the same for all measurements except that different Individuals Databases are used, as will be explained below. In this step, no reasoner libraries are needed, since the related ontology processing steps are all carried out with the Jena2 library [24] [25]. The related measurement results are shown in detail in Appendix F: Measurement Results. The arithmetic mean $\bar{x}$ of the runtime measurements are 1320.3 milliseconds. The arithmetic mean $\bar{x}$ has been introduced in section 6.1.3. However, as mentioned in section 6.1.3, the ontology model created in this step could be loaded once and kept in memory or stored as a whole as long as there are no changes either in the user profile ontology, the context ontology or the Individuals Database. Hence, this measurement step is not that important for the evaluation of our user profile selection mechanism.

From now on, we concentrate on the second measurement step. That is, the forthcoming measurement results shown in this section represent the results for the measurement step 2 of section 6.1.3. In the measurement step 2, first a String object containing the RDF/XML serialisation [82] of the combined ontology model is converted to the required internal model representation of the corresponding ontology reasoner. Afterwards, the current user situation is added to that model, and finally, the corresponding query is created and executed.

6.2.1 Comparison of Reasoner Performance

The measurements for the measurement step 2 described above has been carried out for different ontology reasoning libraries. In particular, we have used Jena2 [24], KAON2 [26], Pellet [30], and FaCT++ [32]. The Pellet reasoner has even been used with three different configurations. These reasoners and their configurations are explained in the following.

Jena2 Reasoner:
Primarily, the Jena2 library is a Java-based library for managing RDF [17] and OWL [20] documents, i.e. for the creation, manipulation, storage and retrieval of RDF and OWL documents. As explained in section 6.1.1, we, too, use this library for these purposes, in particular version 2.5.4 of the Jena2 library. However, the Jena2 library also supports SPARQL queries [23] and includes a rule-based inference engine. For the purpose of reasoning, an inference model (Jena2 InfModel) has to be created that includes the ontology to be queried. In addition, this inference model has to be configured with a reasoner. The corresponding source code can be seen below.

```
Model model = ModelFactory.createDefaultModel();
Reasoner reasoner = ReasonerRegistry.getOWLReasoner();
InfModel infModel = ModelFactory.createInfModel(reasoner, model);
```

In this source code extract, first an empty DefaultModel is created. Afterwards a reasoner is instantiated, in particular the Jena2 OWL Reasoner. Subsequently, an inference model is instantiated and the reasoner and DefaultModel are assigned to the inference model. In the next step, not shown here, first our combined ontology from measurement step 1 is added to the inference model, second the user's current situation is also added afterwards.

Jena2 also provides another possibility for configuring reasoners. In this second possibility, an ontology model specification (Jena2 OntModelSpec) is assigned to the ontology to be queried. Besides the ontology model specification OWL_MEM_RULE_INF for the Jena2 OWL Reasoner introduced above, Jena2 also provides several other ontology model specifications, e.g. the OWL_MEM_TRANS_INF specification for transitive class-hierarchy inference, or the OWL_MEM_MINI_RULE_INF specification for a rule-based reasoner with a subset of OWL rules. By means of this mechanism, also external ontology model specifications can be used. One such example is the ontology model specification of the Pellet reasoner, which is explained later.

Although these two possibilities for configuring a reasoner in Jena2 are documented to be equivalent from a functional point of view, the former requires less execution time, as our experiments have shown. Hence, our measurements are based on the implementation with the source code extract above. Furthermore, our implementation continues with the source code extract below, in which the actual query is executed, and the query result is returned in terms of a Boolean value. The query parameter in the first source code line is a SPARQL ASK query, as introduced in section 5.1.

```
Query sparqlQuery = QueryFactory.create(query);
```

```
QueryExecution queryExec = QueryExecutionFactory.create(sparqlQuery, infModel);
String result = Boolean.toString(queryExec.execAsk());
```

Pellet Reasoner (Configuration 1):
Pellet is an OWL DL (OWL Description Logics) reasoner in Java. It supports the full expressivity of OWL DL. The Pellet version we have used is version 1.5.1. The Pellet library uses the Jena2 library for managing ontologies, which is why we can create a Jena2 DefaultModel at the beginning, as shown below, and subsequently add our combined ontology from measurement step 1, and the user's current situation.

```
Model model = ModelFactory.createDefaultModel();
```

Subsequently, the Pellet reasoner is instantiated, the ontology model is loaded, and the consistency of the model is checked:

```
OWLReasoner reasoner = new OWLReasoner();
reasoner.setDiscardJenaGraph(true);
reasoner.load(model);
KnowledgeBase kb = reasoner.getKB();
kb.isConsistent();
```

Finally, the query is executed, where the queryString parameter is a SPARQL ASK query. If the final results parameter is empty, the result is equivalent to the Boolean value false, otherwise to the Boolean value true. Unfortunately, the used Pellet version does not support the UNION construct used in our SPARQL queries in section 5.2. However, this is not a problem, as the UNION construct expresses a disjunction and hence can be substituted with two SPARQL queries one after another.

```
QueryParser parser = QueryEngine.createParser();
org.mindswap.pellet.query.Query query = parser.parse(queryString, kb);
QueryResults results = QueryEngine.exec(query);
```

Pellet Reasoner (Configuration 2):
In this Pellet configuration, we use the mechanism described in the Jena2 Reasoner section above, in which we assign the Pellet specific ontology model specification (Jena2 OntModelSpec) to the ontology to be queried. The instantiation of the related ontology model looks as follows, where the PelletReasonerFactory.THE_SPEC is the Pellet specific ontology model specification.

```
OntModel model =
ModelFactory.createOntologyModel(PelletReasonerFactory.THE_SPEC);
```

The subsequent processing steps are the same as for the Jena2 Reasoner:

```
Query sparqlQuery = QueryFactory.create(query);
QueryExecution queryExec = QueryExecutionFactory.create(sparqlQuery, model);
String result = Boolean.toString(queryExec.execAsk());
```

One advantage of this configuration is that one does not require detailed programming skills of the Pellet library. Every line of source code is based on the Jena2 library, except that the Pellet specific PelletReasonerFactory.THE_SPEC specification has to be added.

Pellet Reasoner (DIG Configuration):
In this third Pellet configuration, the Pellet reasoner is used via a DIG interface. The DIG interface is an interface that has been specified by the Description Logic Implementation Group (DIG). The DIG interface is a standard for providing access to description-logic reasoning via an HTTP (Hypertext Transfer Protocol)-based interface to a separate reasoning process. In this configuration, the Pellet reasoner is started as a DIG server beforehand and can be accessed via the corresponding IP address and port. When starting the Pellet DIG server in the default configuration, which we have not changed, the Java virtual machine is started with the arguments -Xms30m -Xmx200m. The -Xms30m argument sets the initial amount of memory for the Java virtual machine to 30 MBytes, and the -Xmx200m argument sets the maximum amount of memory for the Java virtual machine to 200 MBytes, compared to 63.5625 MBytes, which is the default configuration on our system as mentioned in section 6.1.1.

The request to the DIG server is done with the Jena2 library, which has to be configured accordingly. In the below example, first a configuration resource is set up to connect to the external reasoner that is accessible at the address localhost and port 8081.

```
        Model cModel= ModelFactory.createDefaultModel();
        Resource conf= cModel.createResource();
        conf.addProperty(ReasonerVocabulary.EXT_REASONER_URL,
cModel.createResource("http://localhost:8081"));
```

In the following steps, a DIGReasoner object is created that can bind an ontology graph to an external reasoner:

```
        DIGReasonerFactory drf= (DIGReasonerFactory)ReasonerRegistry.theRegistry().
getFactory(DIGReasonerFactory.URI);
        DIGReasoner r= (DIGReasoner)drf.create(conf);
```

Next, we create an ontology model specification that includes the DIG reasoner created above:

```
        OntModelSpec spec= new OntModelSpec(OntModelSpec.OWL_DL_MEM);
        spec.setReasoner(r);
        OntModel model= ModelFactory.createOntologyModel(spec);
```

In the next step, not shown here, our combined ontology from measurement step 1 and the user’s current situation have to be added to the created ontology model. The subsequent processing steps are the same as for the Jena2 Reasoner:

```
        Query sparqlQuery = QueryFactory.create(query);
        QueryExecution queryExec = QueryExecutionFactory.create(sparqlQuery, model);
        String result = Boolean.toString(queryExec.execAsk());
```

KAON2 Reasoner:
Primarily, KAON2 is a Java-based infrastructure for managing OWL DL ontologies and other kinds of ontologies. However, it also provides ontology reasoning support. The KAON2 version we have used is the version of January 14, 2008. In order to create the ontology, we have to start by creating a connection and by registering a so-called resolver that provides a physical URI for the ontology:

```
        KAON2Connection connection=KAON2Manager.newConnection();
        DefaultOntologyResolver resolver=new DefaultOntologyResolver();
        connection.setOntologyResolver(resolver);
```

Afterwards, we read the ontology from an input stream (Java InputStream). This time, we can not directly use objects, i.e. ontology models, from the Jena2 library, as the KAON2 library is independent from the Jena2 library. Hence, we first create the combined ontology model with Jena2, add the user's current situation with Jena2, and finally convert the Jena2 ontology model to an input stream (Java InputStream), which can then be processed from the KAON2 library for reading the ontology. The modelStream parameter below represents the ontology model as an input stream.

```
        HashMap hm= new HashMap();
        String key= KAON2Connection.LOAD_FROM_INPUT_STREAM;
        InputStream value= modelStream;
        hm.put(key, value);
        Ontology ontology=connection.openOntology("http://www.ex.de/upos.owl", hm);
```

Subsequently, the reasoner is created, a query object is created, and the query is executed. The queryString parameter represents the corresponding SPARQL ASK query.

```
        Reasoner reasoner=ontology.createReasoner();
        Query myQuery=reasoner.createQuery(new Namespaces(Namespaces.INSTANCE),
queryString);
        myQuery.open();
```

FaCT++ Reasoner (DIG Configuration):
FaCT++ is an OWL DL reasoner, implemented using C++. As well as the Pellet reasoner, it provides a DIG interface. We have used the FaCT++ DIG reasoner, in particular the version 1.1.10, in the same way as the Pellet DIG reasoner. Hence, the source code is the same as in the section for the Pellet Reasoner with DIG configuration, except that it has to be adapted to the appropriate IP address and port.

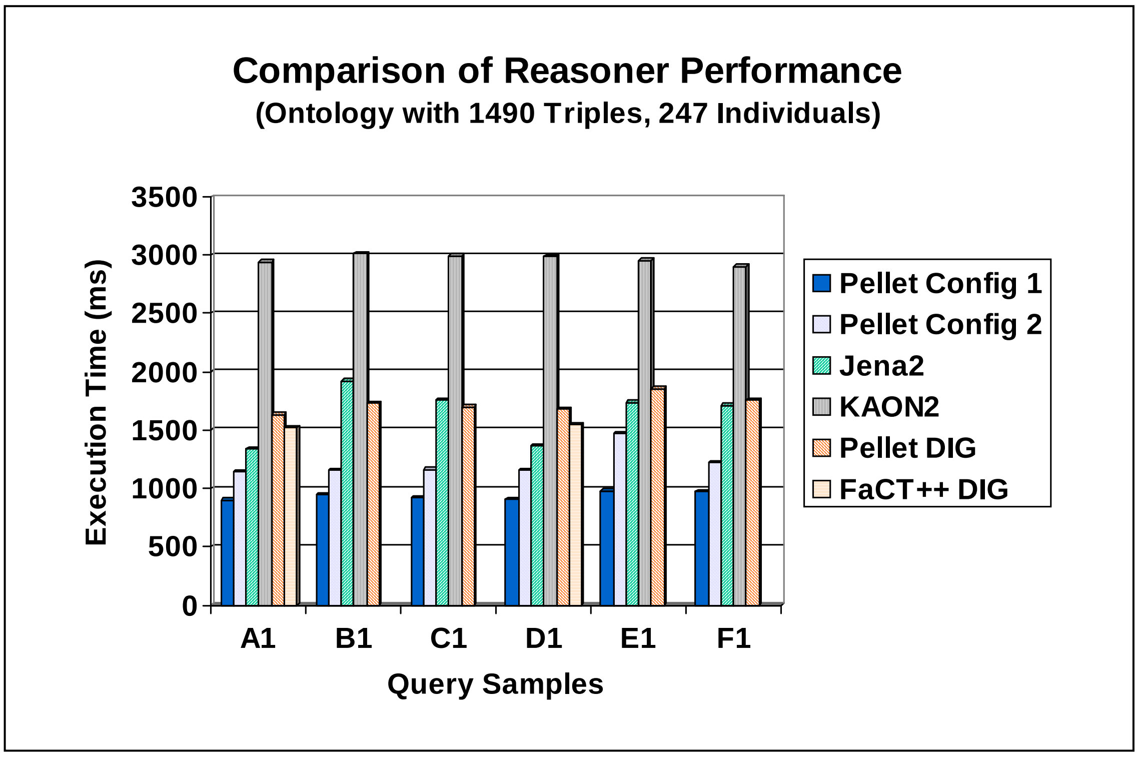


Figure 43: Execution Time for Different Reasoners

As can be seen in Figure 43, there are considerable differences in the execution time between the different reasoners and reasoner configurations. As mentioned in section 6.1.2, the combined ontology, which is queried here, consists of 1490 triples and 247 individuals. The diagram shows the execution time in milliseconds for the complete measurement step 2, see section 6.1.3, for six of the query samples introduced in section 6.1.4. As can be seen, the Pellet reasoner with configuration 1 clearly outperforms all other reasoners and other Pellet reasoner configurations. The second best performance is provided by the Pellet reasoner with configuration 2.

Third is either the Jena2 or the Pellet reasoner with DIG configuration. The FaCT++ DIG reasoner only successfully answers the query samples A1 and D1. This is not surprising, as the FaCT++ reasoner in a first instance supports terminological reasoning [30] [32], also called T-Box (Terminological Box) reasoning. Terminological reasoning asserts facts about concepts (sets of objects) and roles (binary relations). Contrary to this, assertional reasoning, also called A-Box (Assertional Box) reasoning asserts facts about individuals (single objects). A-Box reasoning is neither required in the query sample A1, nor in the query sample D1. As described in section 5.2.1, query sample A1 does not require any reasoning at all, and query sample D1 requires reasoning over the class hierarchy, i.e. requires terminological reasoning. By contract, the execution of the query samples B1, C1, E1 and F1 include reasoning over individuals, i.e. assertional reasoning, and hence can not successfully be answered by FaCT++. Finally, Figure 43 clearly shows that the KAON2 reasoner performs the worst for these query samples.

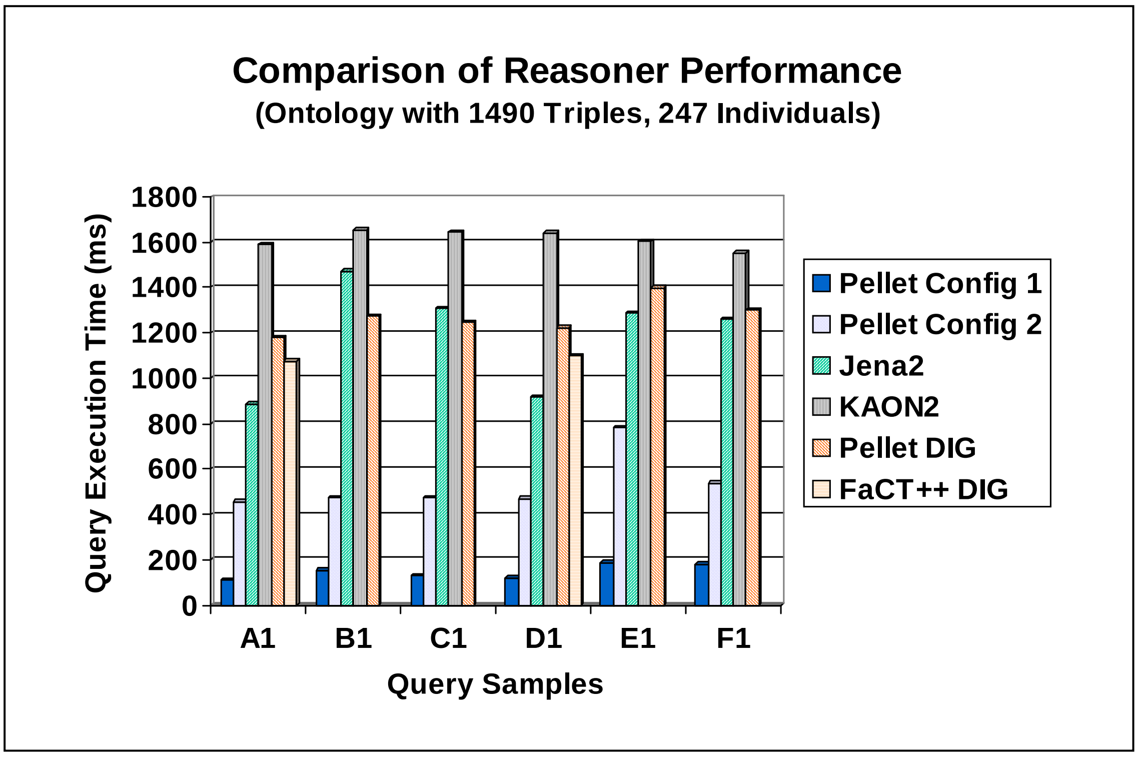


Figure 44: Query-Only Execution Time for Different Reasoners

Figure 44 shows the execution time for the query-only execution. The query-only execution time shown in this diagram does not cover the whole measurement step 2, but only the final query step, i.e. activity 11 (Query ontology model) of the activity diagram in section 5.1. Hence, the other steps of measurement step 2, i.e. activities 4 and 8 to 10 of the activity diagram, as well as reasoner preparation time are not included.

As can be seen in Figure 44, the qualitative results are the same as for Figure 43, i.e. the winner is Pellet with configuration 1, second is Pellet with configuration 2, third are either Jena2 or Pellet with DIG configuration and worst is KAON2. However, the quantitative results differ considerably from that ones in Figure 43. In particular, all reasoners require several hundreds of milliseconds less. The highest performance gain can be documented for the KAON2 reasoner and the Pellet reasoner with configuration 1.

This comparison is interesting in case we assume that several queries have to be carried out one after another. This is usually the case, as we need one query for each situational condition within a user profile, and as a user profile usually has multiple situational conditions. In this case, we do not have to read in and configure the ontology model, include the user’s current situation and instantiate and prepare the corresponding ontology reasoner for each query again and again. These additional steps are only needed in case the user’s situation changes. In this case, the underlying ontology model has to be changed. In particular, the user’s current situation that has been added to the ontology model has to be removed as it is outdated in this case, and the up-to-date user’s current situation has to be added. In addition, already inferred statements may have to be removed from the ontology model.

Hence, only in case the user’s situation changes, some additional execution time has to be considered in addition to the query-only execution time in Figure 44. Obviously, the maximum execution time in this case is the execution time in Figure 43, i.e. the execution time for the complete measurement step 2. Consequently, we could carry out the whole measurement step 2 including the activities 4 and 8 to 11 of the activity diagram in section 5.1 in case the user’s situation changes. For this purpose, we propose to use the Pellet reasoner with configuration 1, based on the measurement results shown in Figure 43. Furthermore, we could carry out the query-only step, i.e. activity 11 of the activity diagram in section 5.1, in case several situational conditions, i.e. queries, have to be checked for the same user situation. Based on the measurement results shown in Figure 44, we again propose to use the Pellet reasoner with configuration 1.

In a realistic scenario, the Pellet reasoner with configuration 1 would need about 900 milliseconds for the preparation of the ontology model, the preparation of the reasoner and the first query after a situation change of a user has been reported to the user profile selection mechanism. Every subsequent query would in this case need about 100 milliseconds in addition, where the number of needed queries is at most the overall number of situational conditions in the corresponding user profile. This could be acceptable for many non-time critical applications. However, the fact that one query needs about 100 milliseconds shows, why it is important to cluster situation-dependent user data based on situational conditions, as we have proposed it with our user profile structure and ontology. The clustering can avoid redundant queries, and hence helps to reduce the overall execution time.

Detailed measurement results, for both Figure 43 and Figure 44, are provided in Table 11 in Appendix F: Measurement Results. These results show the arithmetic mean, the range and the standard deviation as introduced in section 6.1.3. Another observation based on these measurement results is, that the standard deviation and the range between maximum and minimum values is considerably high for the Pellet DIG configuration in comparison to the other reasoners.

The reasons for the differences in execution times between different reasoners can not be answered completely in all cases. For the Pellet reasoner with configuration 1 and configuration 2, exactly the same reasoner is used. As explained above, only the configurations differ slightly. Unfortunately, neither the documentation of the Jena2 library nor the documentation of the Pellet reasoner gives any hints on expected differences between these two configurations.

The reasons for the differences in execution time between the Pellet reasoner and the Jena2 reasoner could be explained as follows: The developers of the Jena2 reasoner in a first instance focus on the management of RDF and OWL documents. Hence, the provided reasoners within the Jena2 library are not in the main scope of developers, are still incomplete and under development. The Pellet reasoner library on the other hand is a library dedicated to ontology reasoning. The developers of the Pellet library are from one of the most experienced and respected research groups in the world with regard to ontology reasoning. This research group has also participated in the specification of the W3C OWL recommendation. For the ontology reasoning tasks in the Pellet library, they use advanced tableaux algorithms [95] for expressive Description Logics, which may not be used within the Jena2 library.

The reasons for the differences in execution time between the Pellet reasoner and the KAON2 reasoner seem to be due to the used type of algorithms. An important focus of the KAON2 reasoner is on scalable and efficient reasoning with ontologies. Contrary to most other currently available reasoners, KAON2 does not implement the tableaux calculus. Rather, reasoning in KAON2 is implemented by novel algorithms which reduce the corresponding knowledge base to a disjunctive datalog program [96].

Finally, there are the differences in execution time between the Pellet reasoner as used with configuration 1 or 2, and the DIG based variants of Pellet and FaCT++. As the differences in execution time can not be caused by the Pellet reasoning process, it must have to do with the fact that the DIG based reasoners communicate via an HTTP based interface. This interface seems to require additional execution time.

Interesting reasoner performance measurements and detailed discussions on ontology reasoning techniques are also presented in [27] and [30]. In [30], the performance for different test cases is compared between different reasoners, particularly for Pellet, Racer Pro [33], and

FaCT++. The reason, why we have not included Racer Pro in our evaluation is that it is a commercial reasoner. In this comparison, on the one hand, loading, consistency checking and classification timings for some well-known OWL ontologies have been carried out for Pellet. On the other hand, classification times for these three reasoners have been compared concerning different ontologies, and query answering has been compared between Pellet and Racer Pro. The related experiments have shown that Pellet is not as efficient as FaCT++ or Racer Pro in T-Box reasoning tasks, but its performance is still competitive when reasoning with large number of individuals. Furthermore, in [27], a comparison between KAON2, Pellet and Racer is presented. The conclusion of this comparison is that KAON2 performs better than or as good as Pellet and Racer for ontologies with rather simple T-Boxes, but large A-Boxes, but worse for ontologies with large and complex T-Boxes.

However, it is difficult to compare the results of the work in [27] and [30] with ours, as the results depend on the characteristics of the ontology, i.e. the used OWL constructs, the number of classes, properties, individuals and triples, the type of reasoning, i.e. terminological or assertional reasoning, as well as the actual query cases. Hence, all these results should be considered collectively in case a reasoner has to be chosen to achieve a certain goal within a certain system. In this sense, our comparison is specific for our user profile selection mechanism, our corresponding ontologies and our approach of querying the ontology with SPARQL queries of the ASK query type.

6.2.2 Comparison of Query Samples

In this section, we present differences concerning execution time between different query samples. As mentioned in section 6.1.4, for each of the six operators equals, isWithin, isConnectedTo, isA, isWithinA and isConnectedToA, we have created three query samples, identifiable through the identifiers A1 to F3.

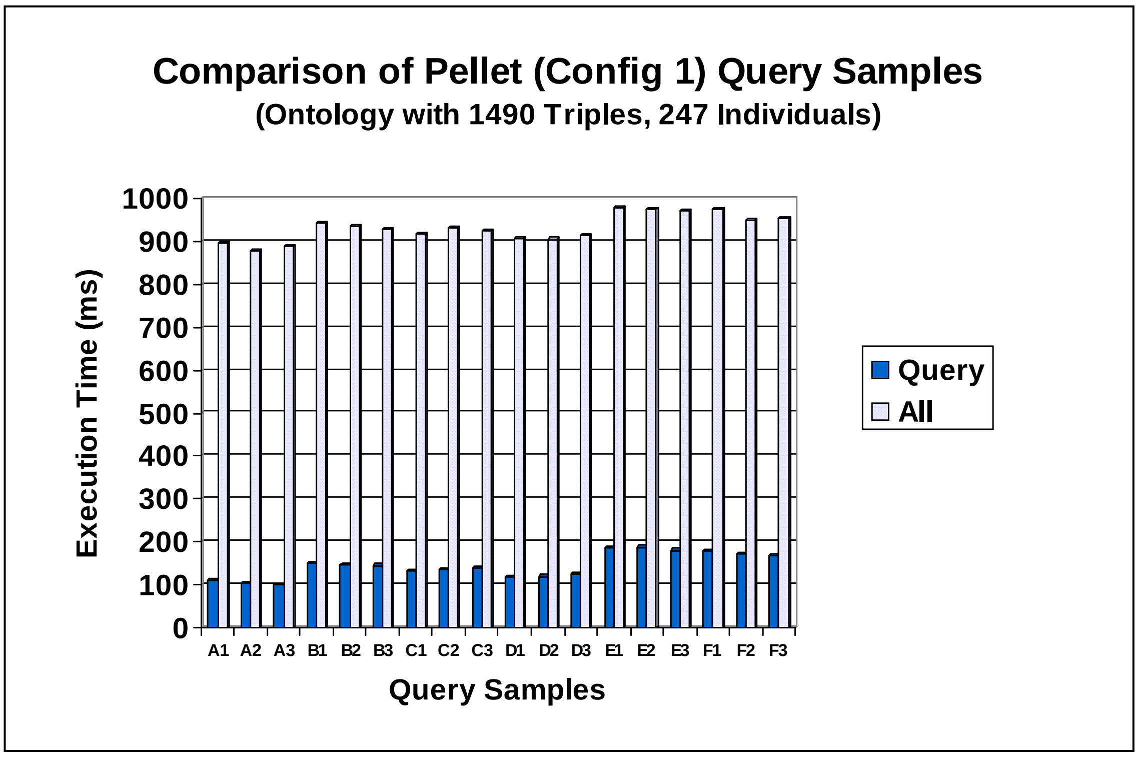


Figure 45: Query Samples for Pellet Reasoner with Configuration 1

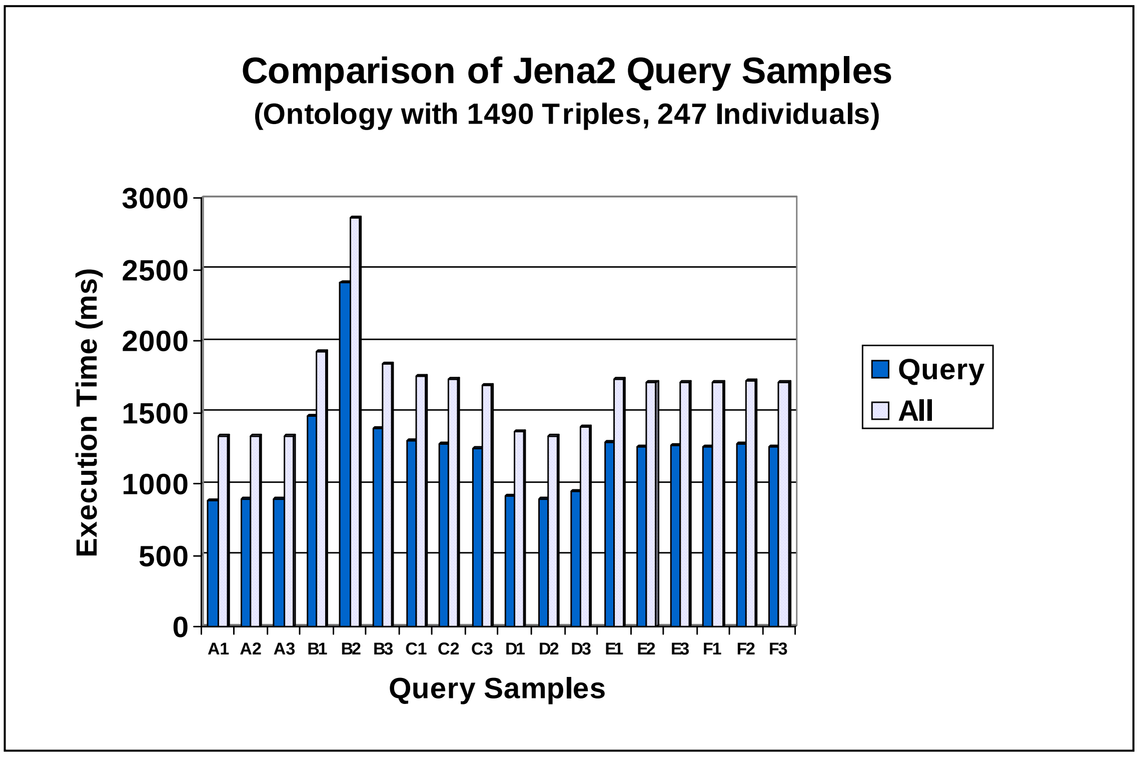


Figure 46: Query Samples for Jena2 Reasoner

Figure 45 shows this comparison for the Pellet reasoner with configuration 1, whereas Figure 46 shows the same comparison for the Jena2 reasoner. In both diagrams, the *All* bar represents the measurement of the complete measurement step 2 as in Figure 43, and the *Query* bar represents the query-only measurement as in Figure 44. As can be seen by comparing both diagrams, the differences in execution times are much more stable for the Pellet reasoner with configuration 1 as for the Jena2 reasoner. The differences in execution time between the query samples of the Pellet reasoner are within 100 milliseconds for both bars. By comparison, the differences in execution time between the query samples of the Jena2 reasoner are up to approximately 1600 milliseconds.

Both diagrams have something in common. In particular, the A and D query samples in both diagrams require less execution time than the B, C, E and F query samples. This may be caused due to the computation of symmetric and transitive relations that do not have to be inferred in the A and D query samples, but in the other query samples. Furthermore, the Jena2 reasoner seems to have a problem with the B2 query sample, in which it requires approximately 1 second more than for most of the other query samples including the query samples B1 and B3, which are similar to B2 in the nature of reasoning that takes place.

The comparisons for KAON2, FaCT++ DIG, Pellet DIG and Pellet with configuration 2 are not shown here. However, their behaviour can also be explained with the above examples. The results for KAON2 are, from a qualitative view, similar to that ones for Pellet in Figure 45. There are only differences up to approximately 100 milliseconds in execution times between all query samples. The same also holds for the FaCT++ DIG reasoner, having in mind that it only supports the query samples A1 and D1 as discussed in the previous section 6.2.1. The Pellet DIG reasoner also shows the same behaviour, but the differences in execution time between the query samples are up to 200 milliseconds, where especially the E and F query samples require more execution time than the other query samples.

The Pellet reasoner with configuration 2 unfortunately reveals partly similar behaviour as the Jena2 reasoner. In particular, the query samples E1, E3, F2 and F3 require up to approximately 300 milliseconds more than all other query samples including E2 and F1, which are similar in the nature of reasoning that takes place. Based on these results, we also propose to use the Pellet reasoner with configuration 1, as it is not only the fastest reasoner as shown in section 6.2.1, but also seems to be very stable in terms of execution time for different query samples.

6.2.3 Comparison of Numbers of Individuals

In this section, we show results of the same measurements as in section 6.2.1, but with bigger Individual Databases, i.e. with more individuals in the Individual Database. In particular, we have created Individuals Databases with about 10 times and with about 100 times as many individuals, i.e. rooms, buildings, etc. as in the standard version in section 6.2.1.

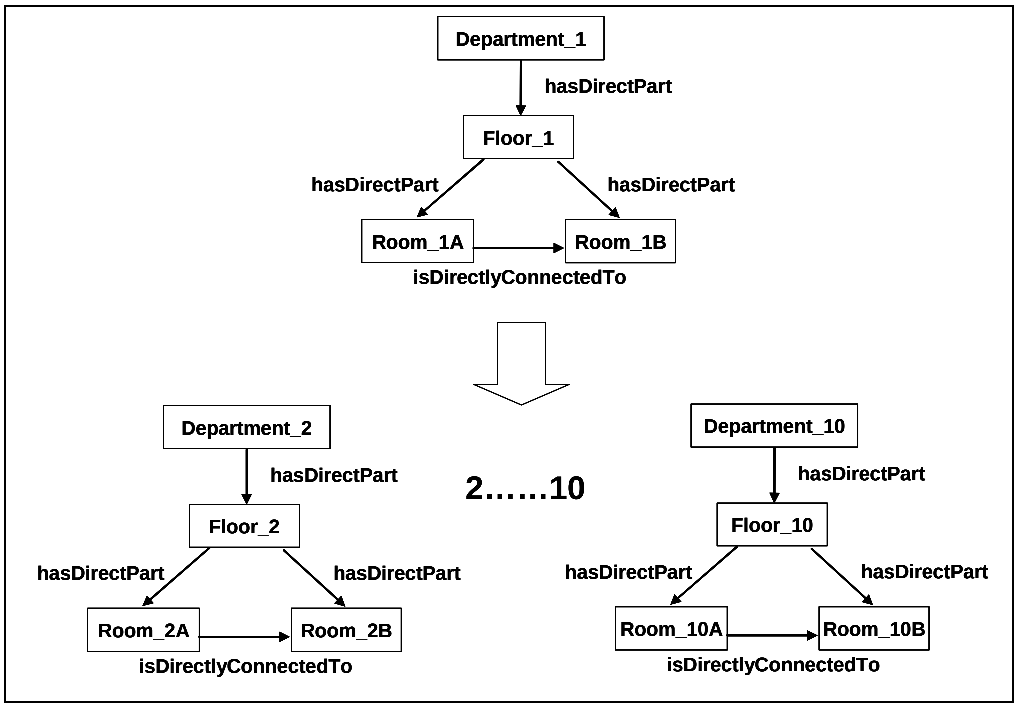


Figure 47: Duplication of and Interrelation between Individuals (Version A)

Figure 47 shows how we have carried out the duplication of the individuals in order to create bigger versions of the Individuals Database. Starting with the example at the top of Figure 47, the same interrelation between individuals has been duplicated 10 times, and the duplicated individuals have been renamed accordingly. Consequently, the resulting Individuals Database contains about 10 times as many individuals. Furthermore, this kind of duplications leads to the fact that each type of location has now 10 times as many instances as before. The distribution of individuals over the location ontology remains the same as before. The same also holds for the relations between individuals, i.e. there are now 10 times as many relations as before. However, the Individuals Database contains 10 independent parts, as there are no relations between individuals of different parts. For instance, there is no relation between the individuals Department_2 and Floor_1. The same has been done for the creation of the Individuals Database with about 100 times as many individuals accordingly. We have called this kind of duplication method Version A, in order to distinguish it from and compare it with another duplication method that is explained in the next section.

Table 6: Characteristics of Individuals Databases (Version A)

Ontology		*# Triples*	*# Classes*	*# Properties*	*# Individuals*
Standard Version	*Individuals Database*	749	88	3	247
	Composed Ontology	1490	207	23	247
10x-A Version	*Individuals Database*	7117	88	3	2056
	Composed Ontology	7861	207	23	2056
100x-A Version	*Individuals Database*	70728	88	3	20146
	Composed Ontology	71472	207	23	20146

The characteristics of the created Individuals Databases with about 10 times (10x-A Version) and with about 100 times (100x-A Version) as many individuals can be seen in Table 6. The Standard Version in this table is the same version as in section 6.1.2, which we have added for comparison purposes. We have counted the triples, classes, properties and individuals in the same way as in section 6.1.2.

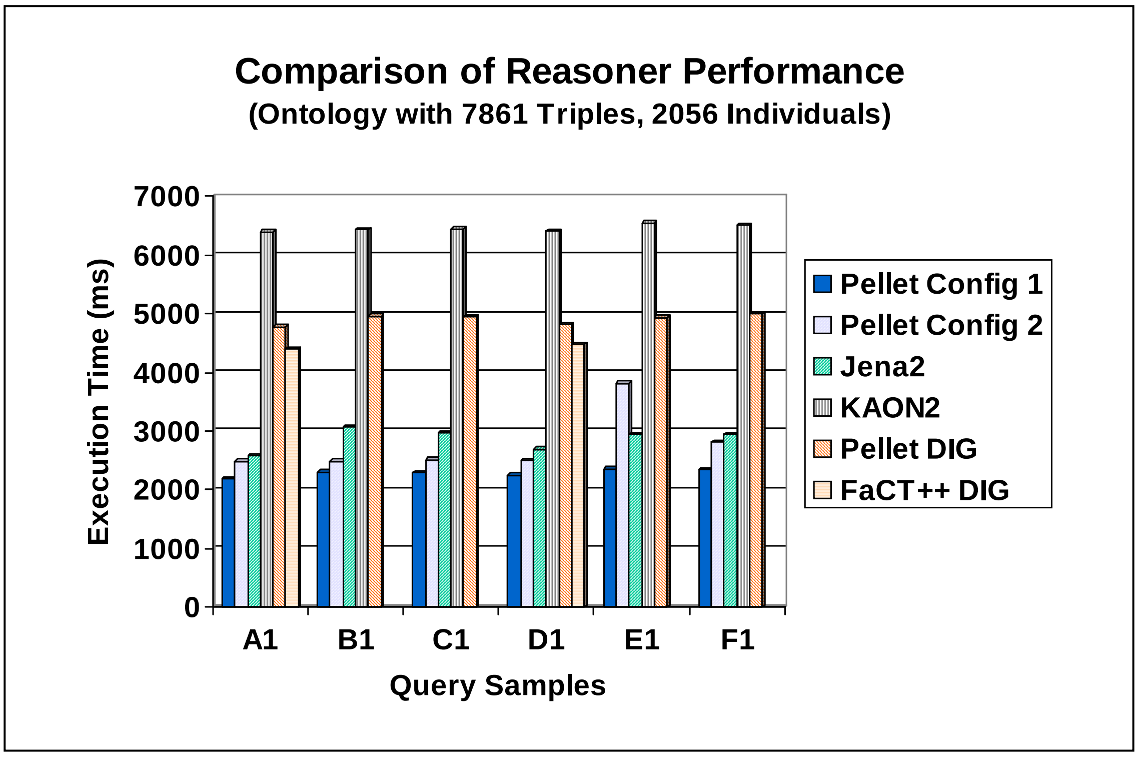


Figure 48: Execution Time for the 10x-A Version

As can be seen in Figure 48, there are also considerable differences in the execution time for the 10x-A Version, similar to that ones in the Standard Version of the Individuals Database in Figure 43. The diagram shows the execution time in milliseconds for the complete measurement step 2 for six of the query samples introduced in section 6.1.4. As can be seen, the Pellet reasoner with configuration 1 performs best, slightly better than the Pellet reasoner with configuration 2 and the Jena2 reasoner. Forth are the two DIG reasoners.

Finally, the KAON2 reasoner performs the worst, as also in Figure 43. Furthermore, for the same reasons as mentioned in section 6.1.4, the FaCT++ DIG reasoner only successfully answers the query samples A1 and D1. The detailed measurement results are shown in Table 12 in Appendix F: Measurement Results.

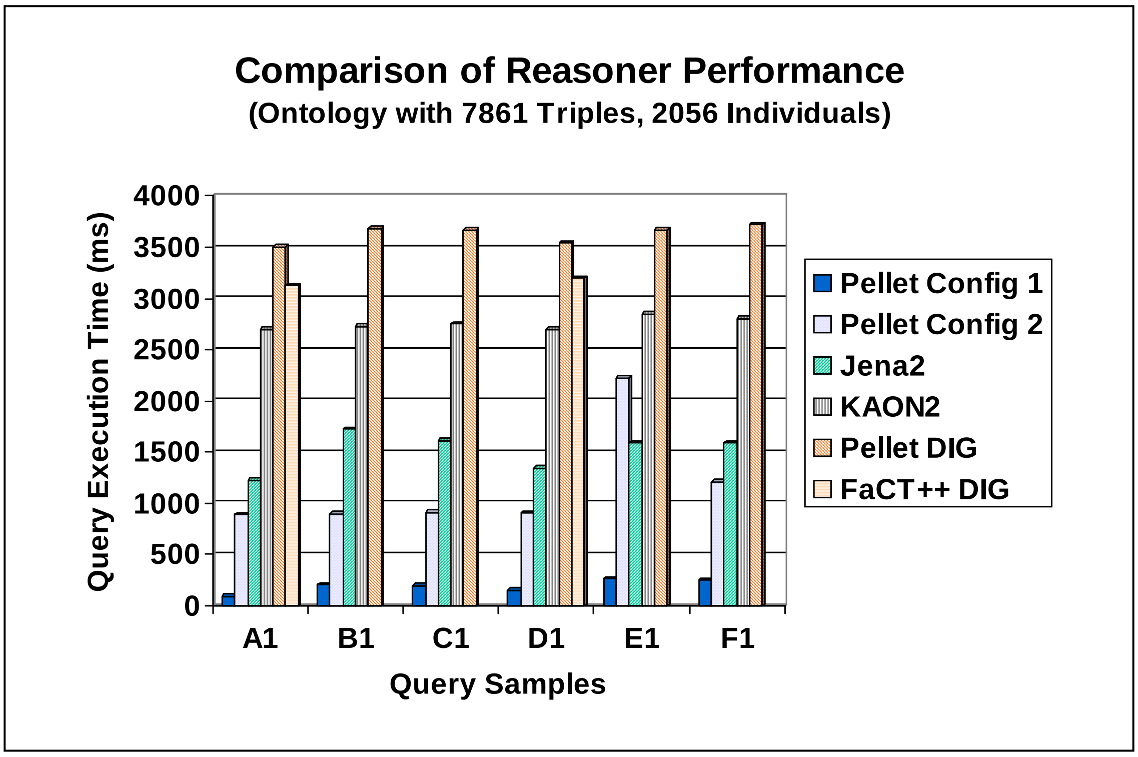


Figure 49: Query-Only Execution Time for the 10x-A Version

Figure 49 shows the execution time for the query-only execution with the 10x-A Version. As can be seen, the query-only execution time for the Pellet reasoner with configuration 1 is only slightly higher than with the Standard Version in Figure 44. All other reasoners require more execution time than with the Standard Version, especially the KAON2 reasoner and the two DIG reasoners. This time, the two DIG reasoners provide the worst performance of all.

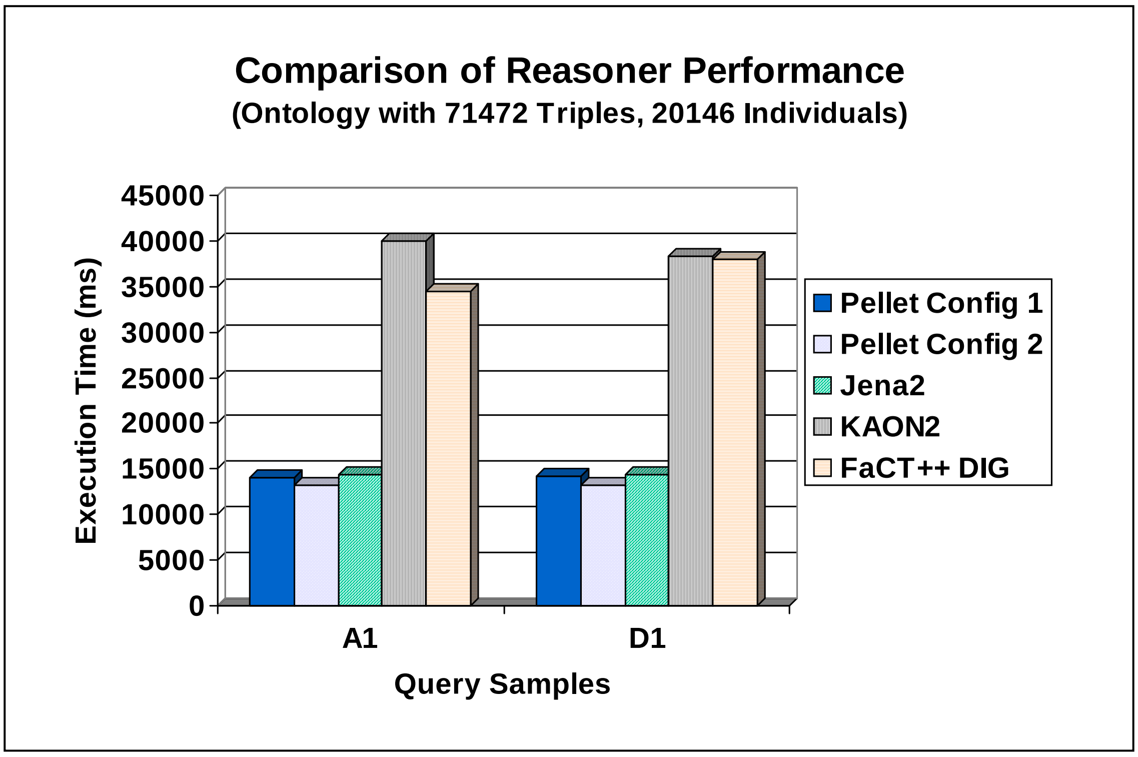


Figure 50: Execution Time for the 100x-A Version

As can be seen in Figure 50, there are also considerable differences in the execution time for the 100x-A Version. The Pellet reasoner with configuration 1, the Pellet reasoner with configuration 2 and the Jena2 reasoner provide the best performance. KAON2 provides the worst performance. The Pellet DIG reasoner on the other hand, has been very unstable. Sometimes it has worked with execution times of about 30 seconds or more, but sometimes an internal server error has been returned. However, there was no reasoner, which could provide results with the default memory configuration for the Java virtual machine described in 6.1.1. Just extending the main memory of our system to 1.5 or 2 GB RAM has not changed this situation either. Hence, for all measurements with the 100x-A Version, we have started the Java virtual machine with the argument -Xmx1600m that increases the available memory for the Java virtual machine. The detailed measurement results are shown in Table 13 in Appendix F: Measurement Results.

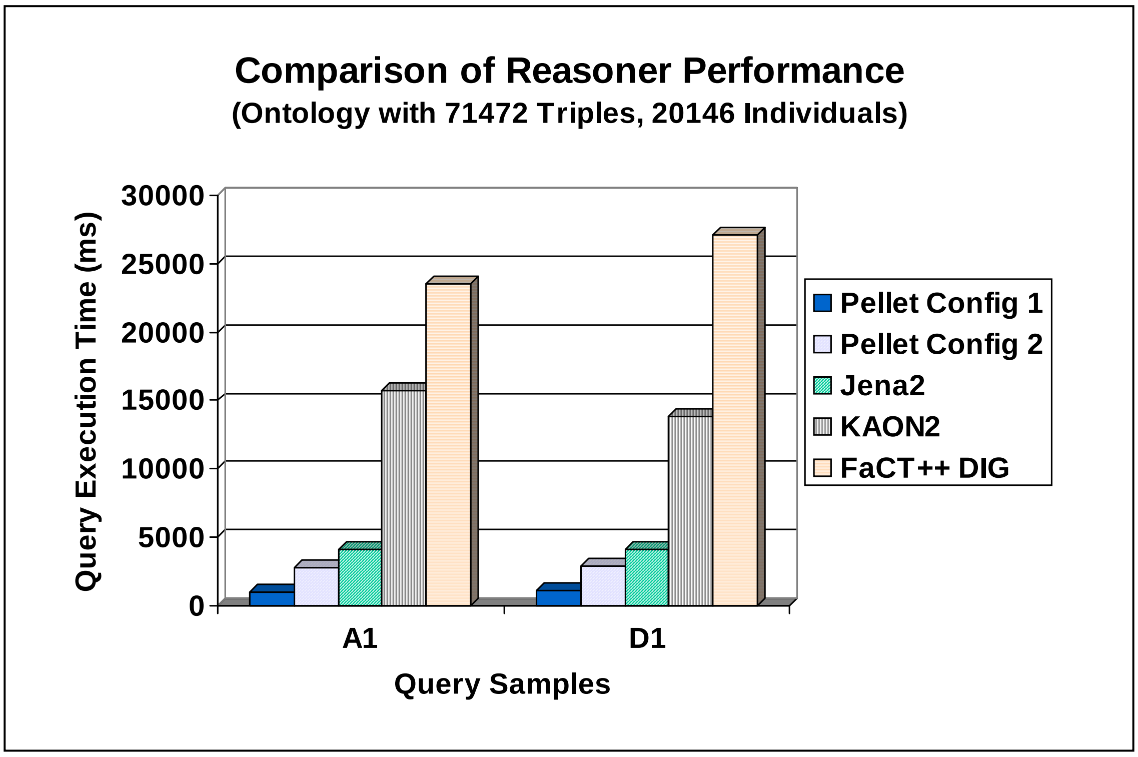


Figure 51: Query-Only Execution Time for the 100x-A Version

Figure 51 shows the execution time for the query-only execution with the 100x-A Version. As can be seen, the query-only execution time for the Pellet reasoner with configuration 1 is the best, followed by the Pellet reasoner with configuration 2, then by the Jena2 reasoner, then by the KAON2 reasoner, and finally by the FaCT++ DIG reasoner.

In summary, we can observe that with more individuals, especially with the 100x-A Version of the Individuals Database, the difference in terms of runtime performance between the reasoners increases. Especially interesting is the query-only evaluation in Figure 49 and Figure 51, which clearly shows the superiority of the Pellet reasoner with configuration 1. The measurement differences between most reasoners for the measurements of the whole measurement step 2 shown in Figure 48 and Figure 50 respectively, minus the query-only measurements is quite small on the other hand. For instance, all reasoners require about 10 seconds or more for the preparation of the ontology model and the preparation of the reasoner with the 100x-A Version. Figure 49 and Figure 51 show that at least for the Pellet reasoners with configuration 1 and 2, the preparation time is a lot higher than the actual query execution time, for both 10x-A Version and 100x-A Version.

For a realistic scenario, the Pellet reasoner with configuration 1 and the 10x-A Version could still be acceptable for many non-time-critical applications. In this case, the preparation of the ontology model, the preparation of the reasoner and the first query would be finished 2.2 seconds after a situation change of a user has been reported to the user profile selection mechanism. Every subsequent query would in this case need about 200 milliseconds in addition, where the number of needed queries is at most the overall number of situational conditions in the corresponding user profile.

6.2.4 Comparison of Interrelations between Individuals

In this section, we show results of the same measurements as in section 6.2.1, but again with different Individuals Databases. In particular, we have created Individuals Databases with about 10 times and with about 100 times as many individuals as in the Standard Version in section 6.2.1. Compared to the Individuals Databases in the previous section 6.2.3, the have chosen another approach for duplicating the individuals, which has lead to a different interrelation between the individuals in the ontology. We also compare the results of these two different duplication approaches in this section. Our motivation for this comparison is to investigate, if the performance of ontology reasoners depends on different interrelations between individuals.

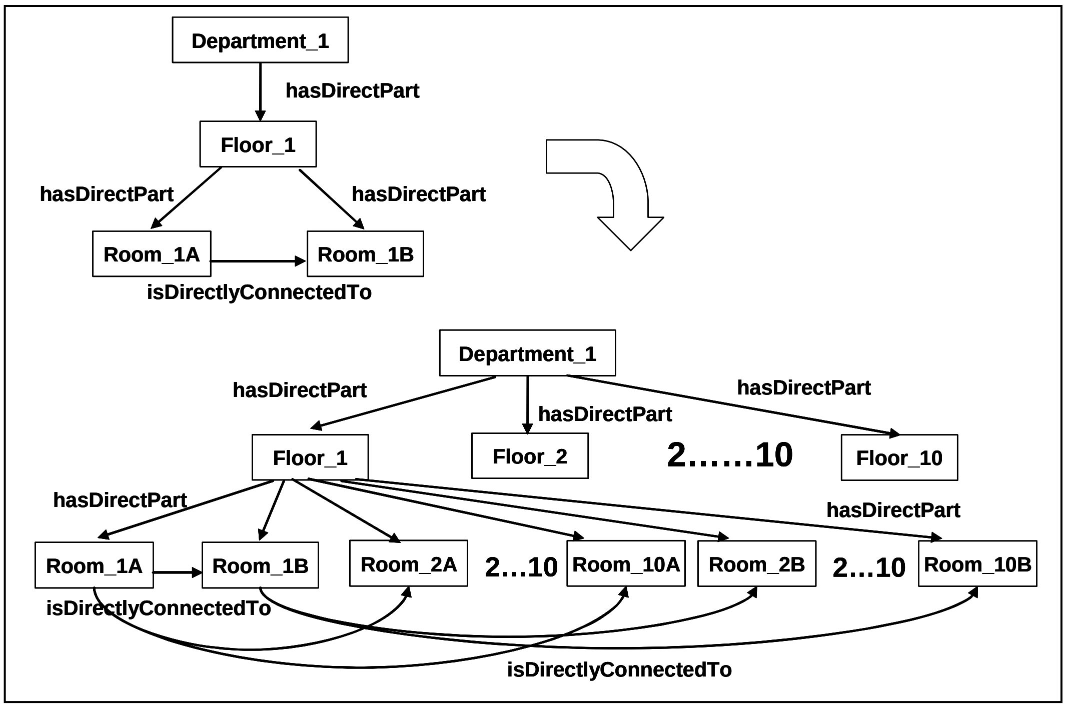


Figure 52: Duplication of and Interrelation between Individuals (Version B)

Figure 52 shows how we have carried out the duplication of the individuals this time in order to create bigger versions of the Individuals Database. Starting with the example at the top of the left-hand side of Figure 52, which is the same example as in the previous section 6.2.3, we have not created 10 independent parts this time. This time, the initial interrelation between individuals has been duplicated in such a way that all resulting individuals are related to the initial individuals. The rule we have applied for this purpose is that we have added 9 additional hasDirectPart relations with 9 additional individuals for each existing hasDirectPart relation. For instance, as the initial Individuals Database contains the hasDirectPart relation between Department_1 and Floor_1, we have added the 9 additional hasDirectPart relations between Department_1 and Floor_2, to Department_1 and Floor_10, where this step also includes the creation of the new individuals Floor_2 to Floor_10. As depicted in Figure 52, the same has also been done for all existing isDirectlyConnectedTo relations. For the creation of the Individuals Database with about 100 times as many individuals, we have done the same accordingly.

As a result, this kind of duplication leads to the same fact as in the duplication approach in section 6.2.3, that each type of location has now about 10 times as many instances as before. That is, also this time, the distribution of individuals over the location ontology remains the same as before. However, this time, as mentioned at the beginning, the individuals are interrelated differently compared to the duplication approach in section 6.2.3. We have called this kind of duplication method Version B, in order to distinguish it from and compare it with the duplication method already introduced in the previous section.

Table 7: Characteristics of Individuals Databases (Version B)

Ontology		*# Triples*	*# Classes*	*# Properties*	*# Individuals*
Standard Version	*Individuals Database*	749	88	3	247
	Composed Ontology	1490	207	23	247
10x-B Version	*Individuals Database*	7013	88	3	2056
	Composed Ontology	7757	207	23	2056
100x-B Version	*Individuals Database*	69316	88	8	20776
	Composed Ontology	70060	207	23	20776

The characteristics of the created Individuals Databases with about 10 times (10x-B Version) and with about 100 times (100x-B Version) as many individuals can be seen in Table 7. The Standard Version in this table is the same version as in section 6.1.2, which we have added for comparison purposes. We have counted the triples, classes, properties and individuals in the same way as in section 6.1.2. The number of individuals and triples for the B Versions of the Individuals Databases is not exactly the same as for the A Versions, but almost the same.

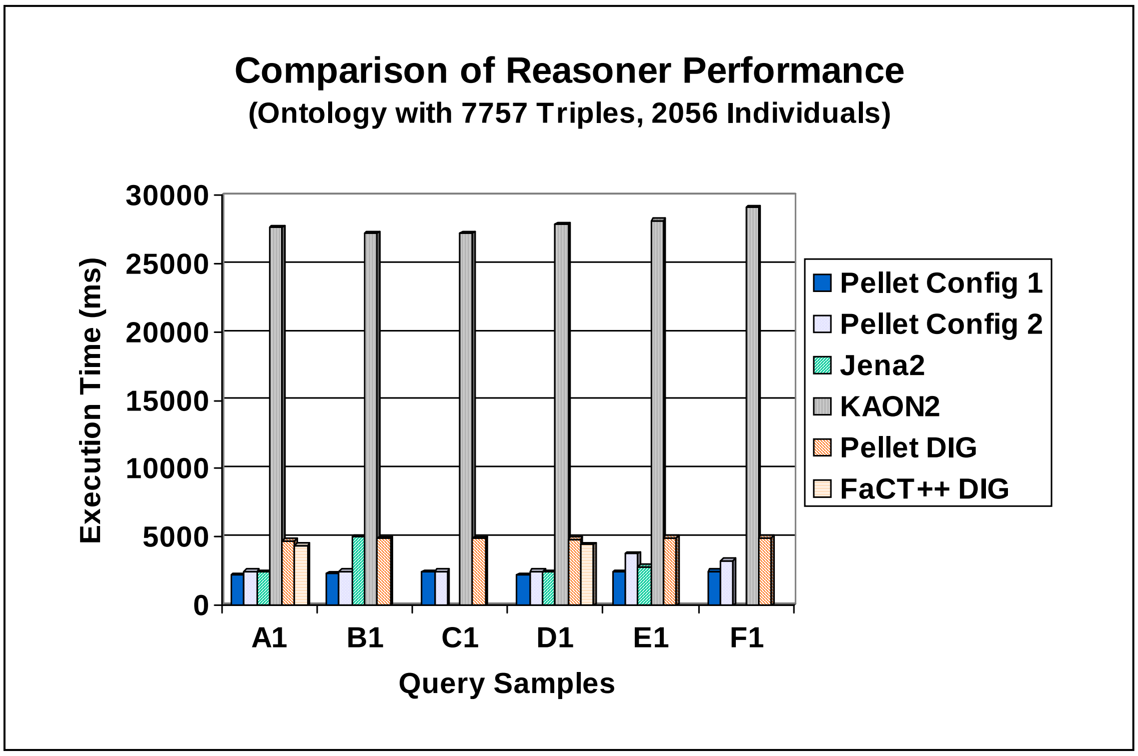


Figure 53: Execution Time for the 10x-B Version

Figure 53 shows the execution time in milliseconds for the complete measurement step 2 for six of the query samples introduced in section 6.1.4. As can be seen in the diagram, there are considerable differences in the execution time for the 10x-B Version. Whereas the Pellet reasoner with configuration 1, the Pellet reasoner with configuration 2 and the Jena2 reasoner have similar execution time and provide the best performance, the two DIG reasoners and the KAON2 reasoner perform worse. The KAON2 reasoner even needs approximately 11 times as much time as the best reasoners in this comparison. For the same reasons as mentioned in section 6.1.4, the FaCT++ DIG reasoner only successfully answers the query samples A1 and D1. The Jena2 reasoner on the other hand, is not able to successfully execute the query cases C1 and F1 due to the Java exception java.lang.OutOfMemoryError. The detailed measurement results are shown in Table 14 in Appendix F: Measurement Results.

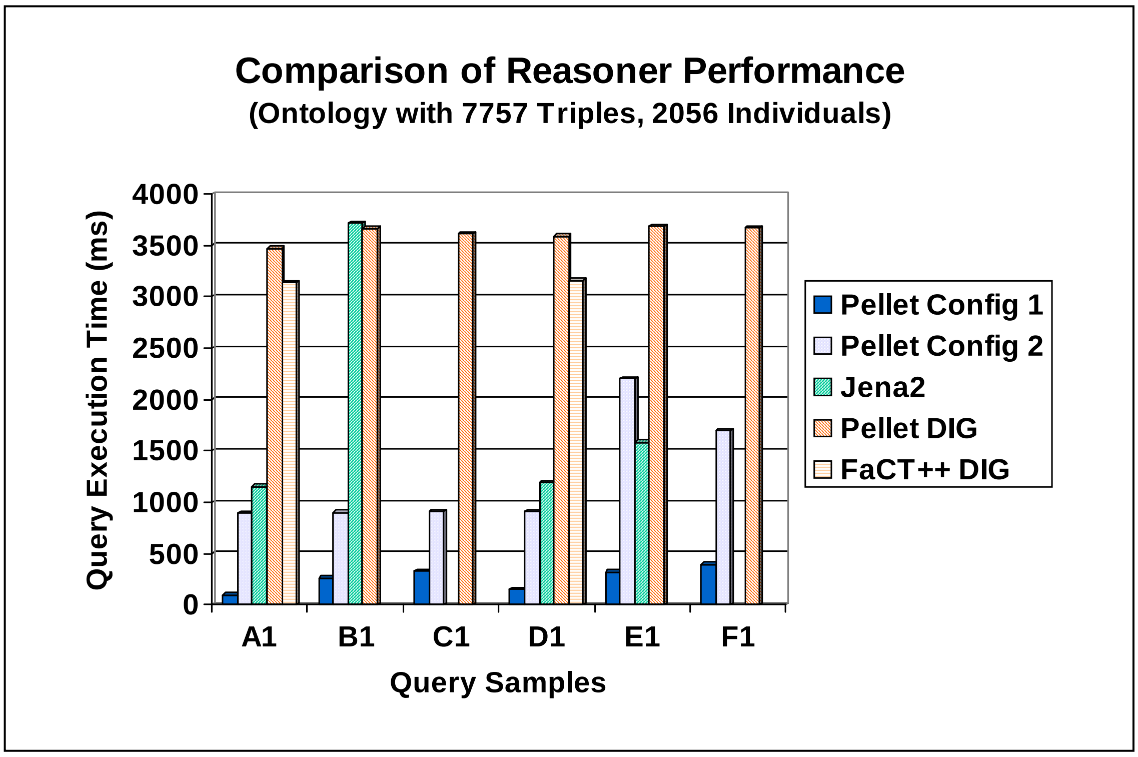


Figure 54: Query-Only Execution Time for the 10x-B Version

Figure 54 shows the execution time for the query-only execution with the 10x-B Version. As can be seen, the query-only execution time for the Pellet reasoner with configuration 1 is only slightly higher than with the Standard Version in Figure 44. All other reasoners require several 100 milliseconds or even several seconds more execution time than with the Standard Version. The results for the KAON2 reasoner are not shown in the diagram for the purpose of better readability. The query-only execution time for the KAON2 reasoner is about 3 to 4 seconds less than the complete execution time, i.e. about 24 seconds for all the query samples.

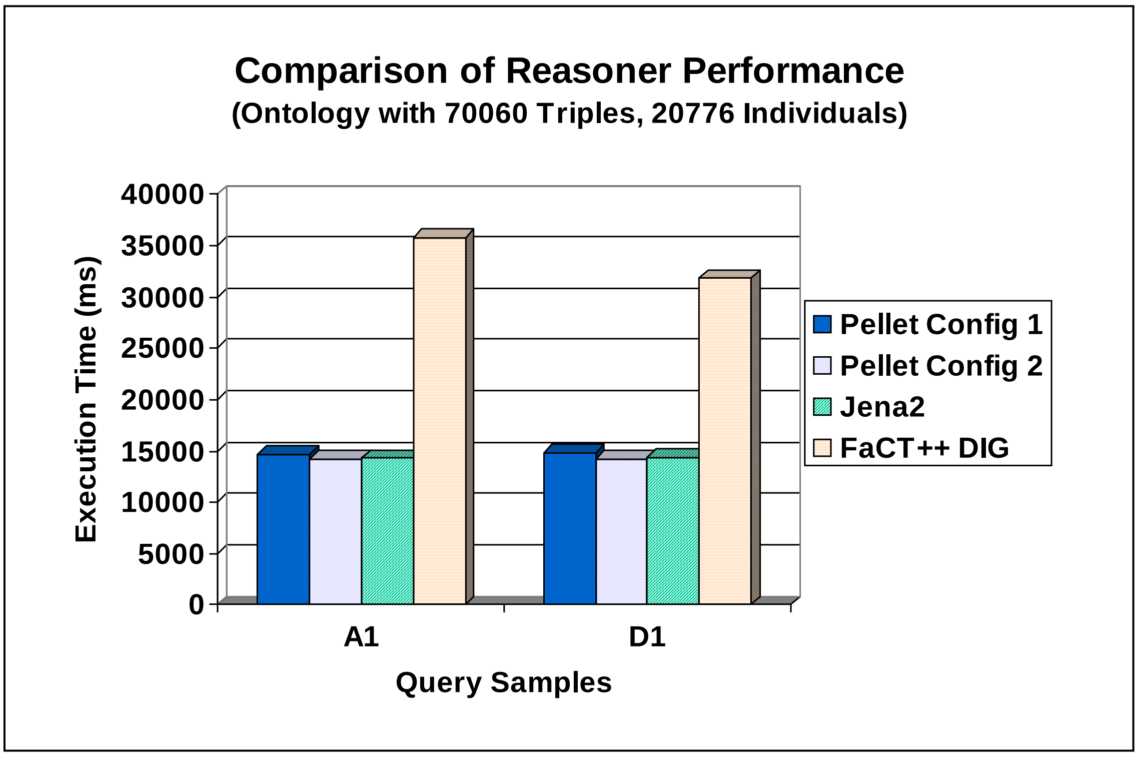


Figure 55: Execution Time for the 100x-B Version

As can be seen in Figure 55, there are also considerable differences in the execution time for the 100x-B Version. The Pellet reasoner with configuration 1, the Pellet reasoner with configuration 2 and the Jena2 reasoner provide the best performance. The Pellet DIG reasoner has been very unstable, as for the 100x-A Version in the previous section. Sometimes it has worked with execution times of about 30 seconds or more, but sometimes an internal server error has been returned. The KAON2 reasoner on the other hand could not provide an answer within 60 seconds, after which we cancelled the query. As for the 100x-A Version, there was no reasoner, which could provide results with the default memory configuration for the Java virtual machine described in 6.1.1. Hence, for all measurements with the 100x-B Version, we have started the Java virtual machine with the argument -Xmx1600m, as for the 100x-A Version. The detailed measurement results are shown in Table 15 in Appendix F: Measurement Results.

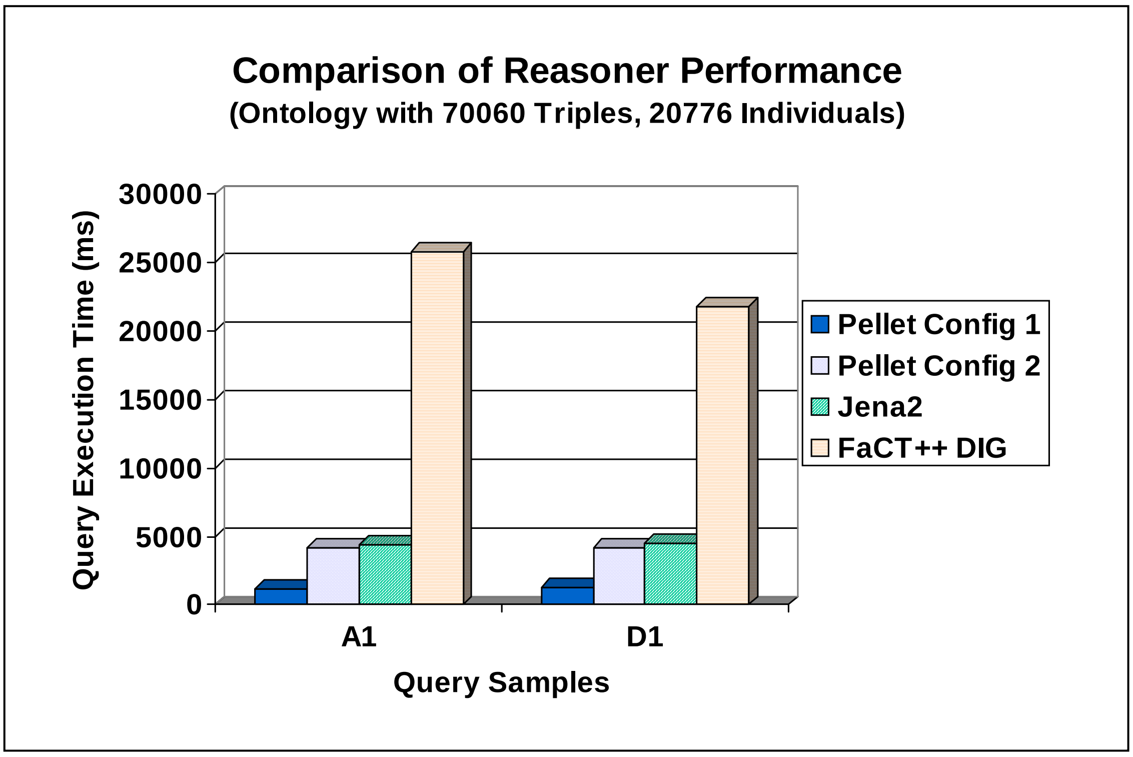


Figure 56: Query-Only Execution Time for the 100x-B Version

Figure 56 shows the corresponding execution time for the query-only execution with the 100x-B Version. As can be seen, the query-only execution time for the Pellet reasoner with configuration 1 is the best, followed by the Pellet reasoner with configuration 2, then by the Jena2 reasoner, and finally by the FaCT++ DIG reasoner.

In summary, we can observe that most reasoners have the same or at least similar behaviour for both duplication approaches. Hence, for these reasoners, the way how individuals are interrelated with each other seems not so relevant for our query samples. However, KAON2 does not show the same behaviour, as is shown later.

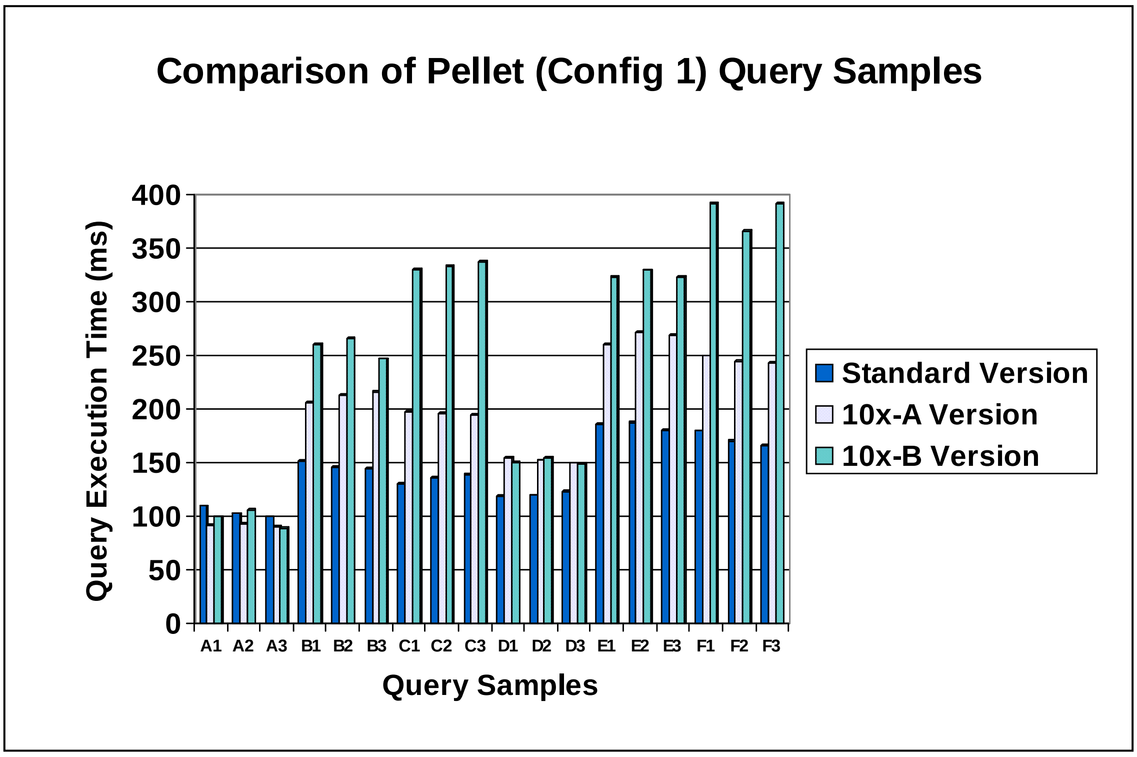


Figure 57: Query Samples for Pellet Reasoner with Configuration 1

Figure 57 depicts the query-only performance of the Pellet reasoner with configuration 1 for the Standard Version, the 10x-A Version, and the 10x-B Version of the Individuals Database. As can be seen, the difference in runtime is insignificant for the A and D query samples. However, it also shows that for most other query samples, the execution time is slightly higher for the 10x-B Version compared to the 10x-A Version. This behaviour is much more pronounced for the KAON2 reasoner, as can be seen below.

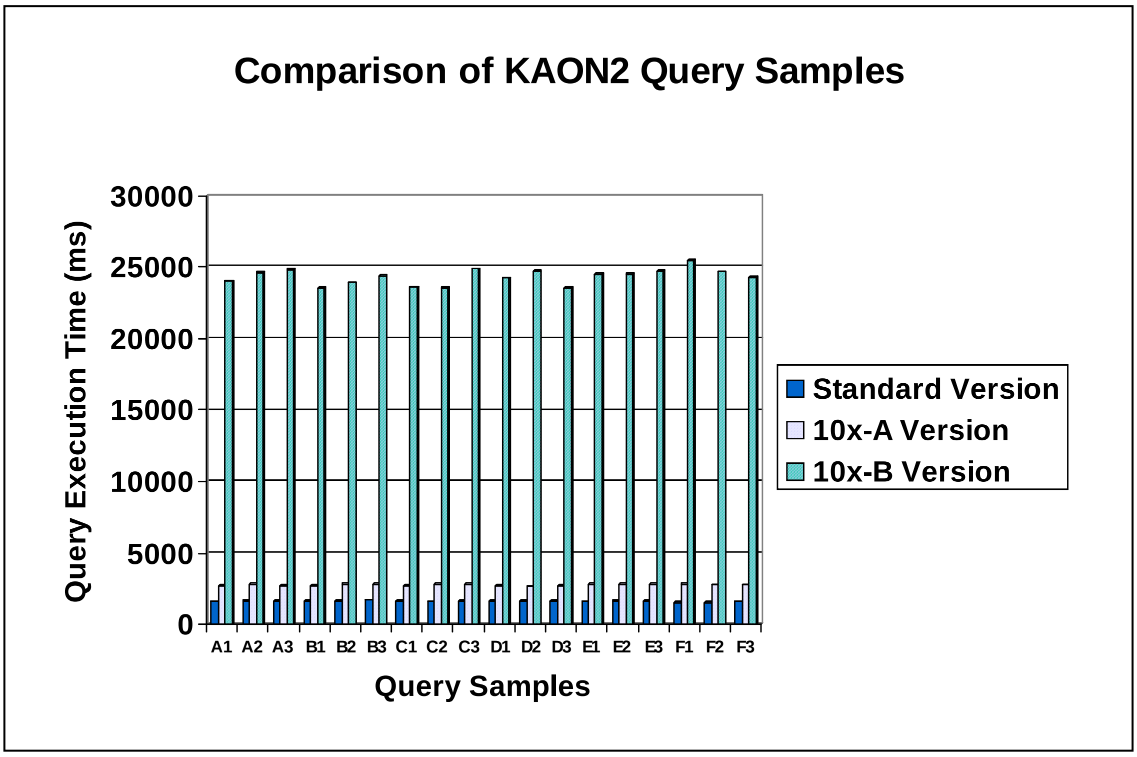


Figure 58: Query Samples for KAON2 Reasoner

Figure 58 depicts the same comparison as Figure 57, but for the KAON2 reasoner. Obviously, there are huge runtime differences depending on the interrelation between the individuals. Hence, we can not recommend this reasoner for our user profile selection mechanism as the content of the Individuals Database is user-specific. As explained in section 5.1.1, the Individuals Database includes the locations the user often visits and stays at, such as her home, the premises in which she works, and the environment in which she lives. Hence, the user-specific Individuals Database could sometimes be similar to that one in the 10x-A Version, and sometimes to that one in the 10x-B Version.

6.2.5 Comparison of User Profile Selection Approaches

In this section, we compare the results of our ontology-based user profile selection approach shown in the previous section with results of other approaches. In particular, we wanted to investigate, whether the use of classifications instead of ontologies decreases runtime. Hence, we have created classifications by changing the ontologies already introduced in this document. For this purpose, we have taken the user profile ontology, the context ontology and the different version of the Individuals Databases and have removed all property specifications and all properties between individuals. As a result, we have got the classifications shown in Table 8. The number of classes is exactly the same as for the ontology version, whereas the number of properties is zero now. The number of individuals is almost the same as for the ontology versions, see Table 6 and Table 7. However, the number of triples is considerably less than for the ontology versions, as all relations between individuals have been dropped.

Table 8: Characteristics of Classifications

Classification		*# Triples*	*# Individuals*
Standard Version	*Individuals Database*	259	247
	Composed Classification	814	247
10x Version	*Individuals Database*	2131	2119
	Composed Classification	2687	2119
100x Version	*Individuals Database*	20851	20839
	Composed Classification	21407	20839

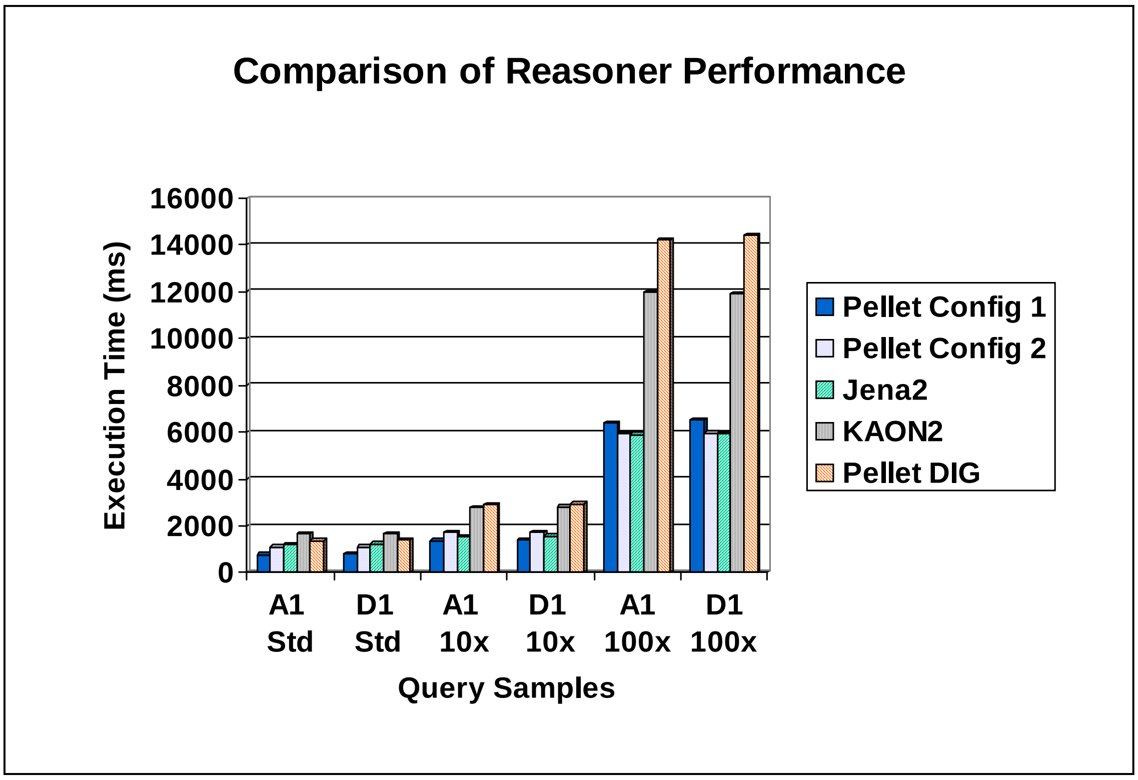


Figure 59: Execution Time for Classification Versions

Figure 59 depicts the results for the overall execution time with the standard (std) classification version, and the 10x and 100x classification version. The use of classifications and reasoning with classifications does not support the B, C, E and F query samples. This is because these samples reason over transitive and symmetric properties, which are not available anymore in the classifications. Hence, only the A and D query samples are supported and shown in Figure 59. Again, the winner in terms of runtime performance is the Pellet reasoner with configuration 1, except for the 100x Version. The worst performance is provided by the KAON2 reasoner and the Pellet DIG reasoner. The detailed measurement results are shown in Table 16, Table 17 and Table 18 in Appendix F: Measurement Results. For the 100x Version, the Jena2 reasoner and the KAON2 reasoner worked with the default memory configuration for the Java virtual machine as described in section 6.1.1. For the other reasoners, we have started the Java virtual machine with the parameter -Xmx1600m, as done for the 100x-A and 100x-B ontology versions in section 6.2.3 and 6.2.4.

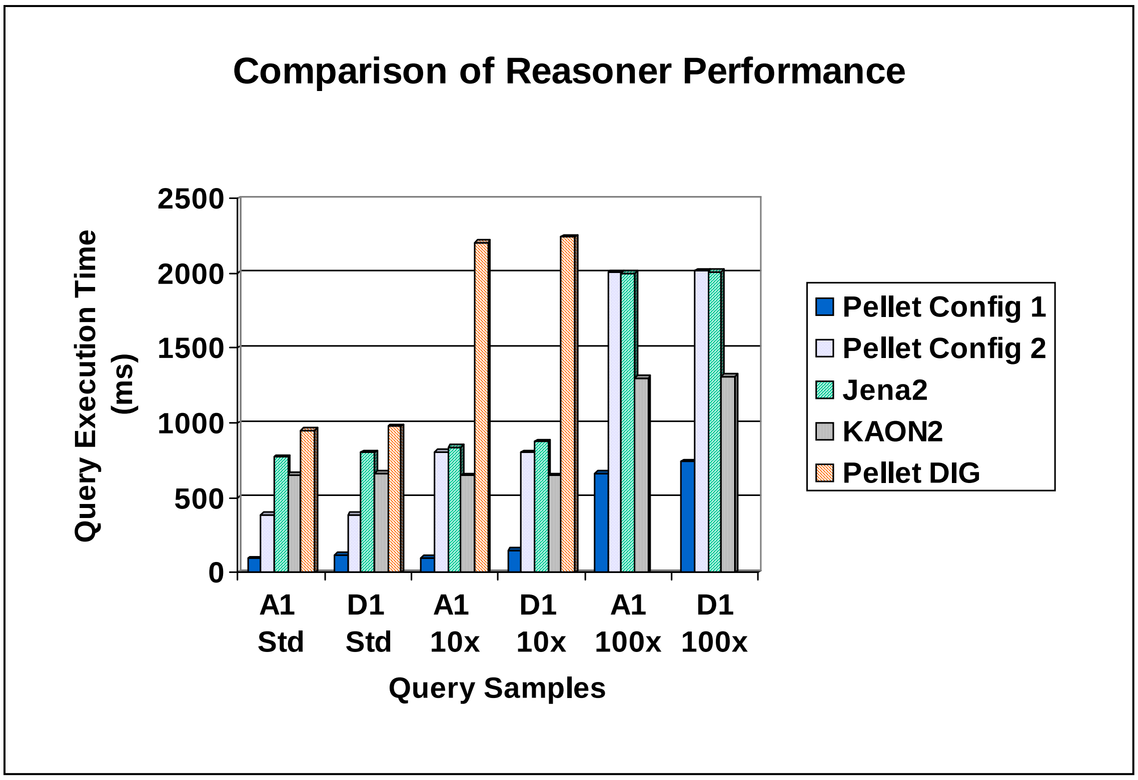


Figure 60: Query-Only Execution Time for Classification Versions

Figure 60 depicts the same comparison as in Figure 59, but for the query-only execution time. This time, the Pellet reasoner with configuration 1 clearly outperforms the other reasoners. For the purpose of readability we have not shown the runtime results for the Pellet DIG reasoner for the query samples A1 and D1 with the classification 100x Version. The results for these two measurements are 14.217 seconds and 14.420 seconds respectively.

Table 9: Comparison of Supported Functionality

Approach ***Operator***	***Default***	***Classification-based***	***Ontology-based***
equals	Yes	Yes	Yes
equals (extended)	No	Yes*	Yes
isA	No	Yes	Yes
isA (extended)	No	Yes*	Yes
isWithin	No	No	Yes
isWithinA	No	No	Yes
isConnectedTo	No	No	Yes
isConnectedToA	No	No	Yes

Table 9 shows the comparison of three user profile selection approaches in terms of supported functionality. As described in the previous sections, our ontology-based approach supports all the introduced operators. The classification-based approach introduced in this section does not support all operators. It only supports the equals operator for checking equality between individuals and the isA operator for checking specialisation and generalisation of classes. Depending on whether we enable the OWL constructs owl:sameAs and owl:equivalentClass in the classification-based approach, the extended equals operator and the extended isA operator are also supported.

The default approach represents the fastest way to process an RDF document. In this approach, the user profile selection mechanism receives the user's current situation as an RDF document from the Context Client Module as is the case for the ontology-based and the classification-based approach, and as is described in section 5.1.1. However, this time, no other ontologies are loaded. That is, we do not load the user profile ontology, the context ontology and the Individuals Database. Instead, we directly query the RDF document including the user's current situation with the corresponding SPARQL query. Obviously, the processing step can not include any ontology reasoning this time, as there is no underlying ontology, i.e. no underlying classification and no relationship information. Hence, the default approach only supports the equals operator for checking equality of individuals. Our motivation to compare it with our ontology-based approach and the classification-based approach is to compare the runtime difference between the fastest way to process and query a small RDF document and the more complex way to query and reason over an ontology.

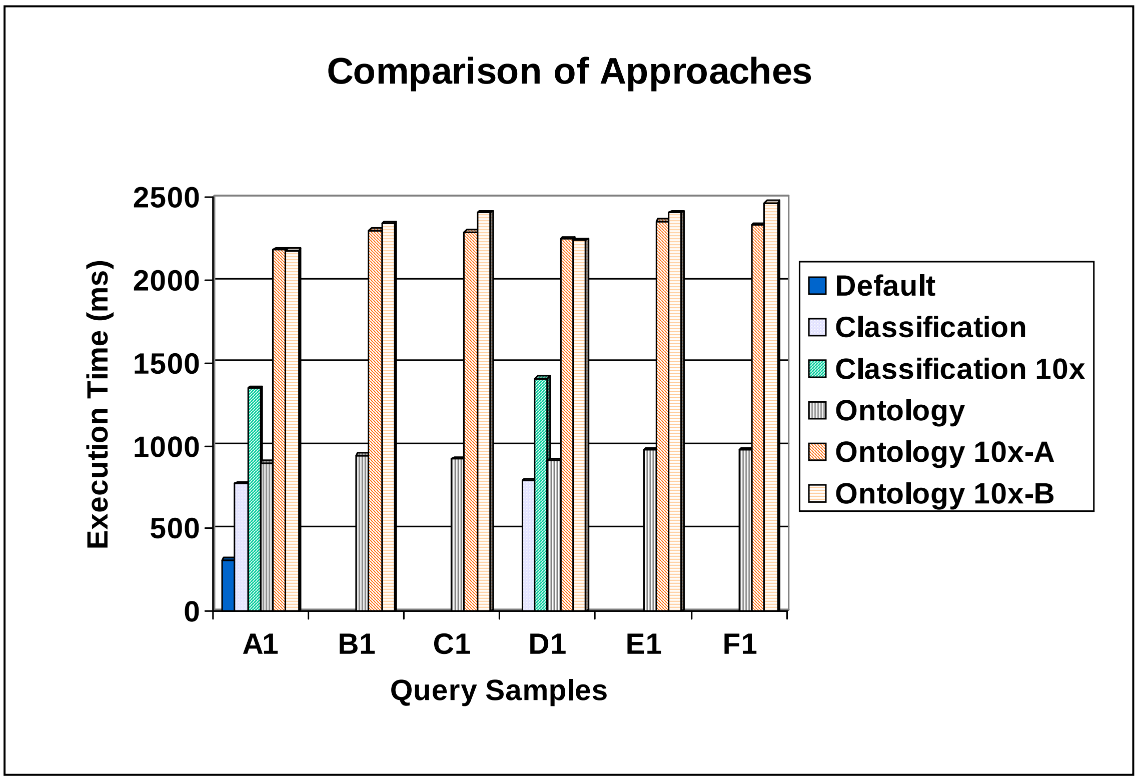


Figure 61: Execution Time of User Profile Selection Approaches

Figure 61 shows the result for this comparison for the overall execution time, i.e. the execution time for the whole measurement step 2 as described in section 6.1.3. As we have expected, the execution time for the default approach is less than for the other approaches, but still needs about 300 milliseconds. The other results shown in Figure 61 represent the fastest reasoner in each case based on the results of the previous sections. In fact, the Pellet reasoner with configuration 1 was the fastest reasoner for each case. Hence, this diagram shows the comparison of the minimum execution times for each approach.

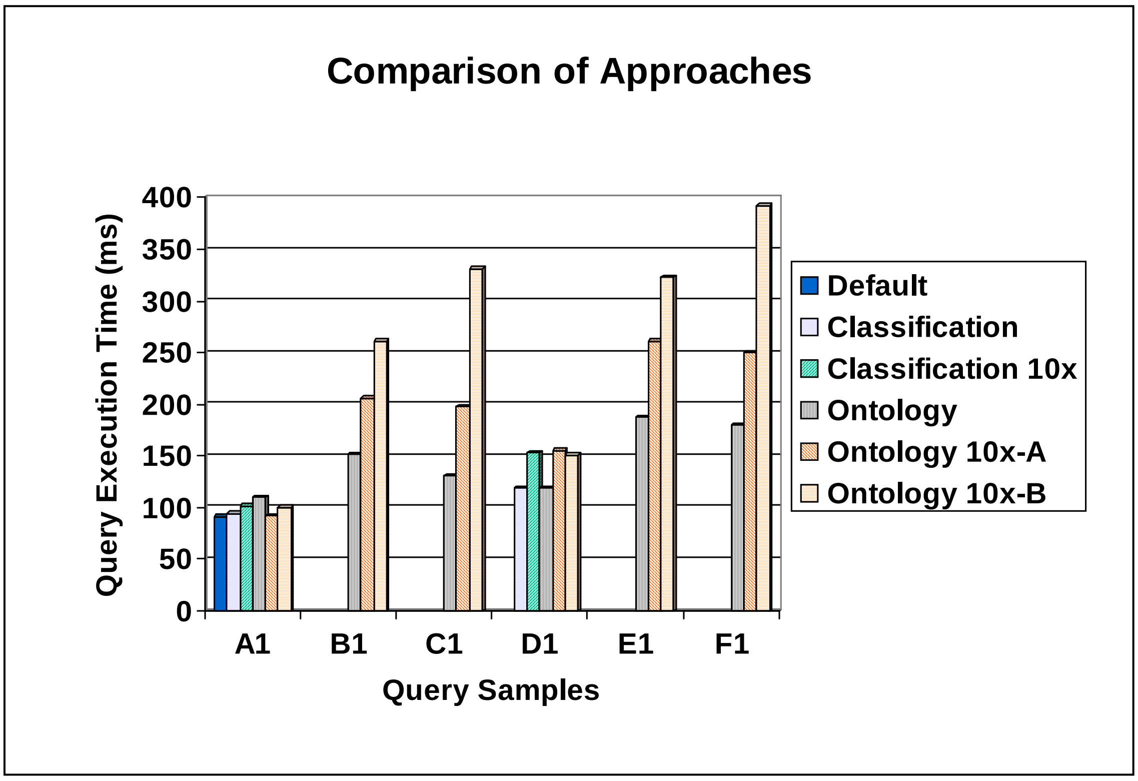


Figure 62: Query-Only Execution Time of User Profile Selection Approaches

Figure 62 depicts the same comparison as Figure 61, but for the query-only execution time. As can be seen, the query-only execution time for the ontology-based and the classification-based approaches, even for the versions with 10 times as many individuals, are almost as fast as for the default approach.

In summary, for the best performing reasoner, i.e. the Pellet reasoner with configuration 1, we hence can observe that the query-only execution time for the ontology-based approach is about as fast as for the query of a simple RDF document in the default approach. However, it increases for some query samples, particularly for the B, C, E and F query samples, which are not supported by the default approach on the other hand. Comparing the results of Figure 62 with those of Figure 61, it becomes clear that the predominant part of the execution time concerns the preparation of the ontology model and the preparation of the reasoner. However, whereas the query-only execution time has to be considered and added up for every situational condition in the queried user profile, the preparation of the ontology model and the preparation of the reasoner has to be done only once for each user profile selection process. The preparation of the ontology model and the reasoner only has to be done at the beginning of the user profile selection process, i.e. in case the user's situation changes and a new user profile selection process is started.

6.3 *Summary*

In this chapter the evaluation of our user profile selection approach in terms of runtime performance and supported functionality has been shown. In the first part of this chapter, we have described the setup for our user profile selection mechanism and the corresponding runtime measurements. We have described the system environment of the user profile management framework, the settings for the Java virtual machine, the software libraries used for the management of RDF documents, and the software libraries used for ontology reasoning tasks.

Furthermore, we have depicted the characteristics of our user profile ontology, our context ontology and our Individuals Database in terms of number of triples, number of classes, number of properties, and number of individuals. Finally, we have also introduced the different measurement steps, the measurement procedure and the measurement samples. For each of the introduced operators in section 4.3.1, we have created and explained three measurement samples. The detailed measurements results are provided in Appendix F: Measurement Results, showing the arithmetic mean, the range between maximum and minimum value, and the standard deviation.

It should be stressed that the results of the evaluation of the user profile selection mechanism, summarised below, are biased due to all the mentioned reasons: choice of hardware and software environment as described in section 6.1.1, implementation approach as described in section 3.2.1 and 5.1.1, design of ontologies as described in chapter 4 and dimension of database as described in section 6.1.2. The outcome of this is that there may me a significant opportunity for improving the performance compared to the results described.

The second part of this chapter contains the actual measurement results for the following runtime comparisons:

- Comparison of reasoner libraries
- Comparison of query samples
- Comparison of numbers of individuals
- Comparison of interrelations between individuals
- Comparison of user profile selection approaches

The comparison of reasoner libraries has shown that there are considerable differences in execution time between different reasoners. For our user profile selection mechanism, the Pellet reasoner with configuration 1 has clearly outperformed the other reasoner libraries for both the query-only measurement, as well as the whole measurement step 2. The latter one in addition includes the preparation of the ontology model and the reasoner. By comparison, the KAON2 reasoner performed the worst for both cases.

The comparison of query samples has also shown that different reasoners display different behaviour for different query samples. The range between the maximum and minimum value for the execution time of all query samples has been about 100 milliseconds for the Pellet reasoner with configuration 1. By comparison, this range has been about 1600 milliseconds for the Jena2 reasoner. In summary, the Pellet reasoner with configuration 1 was the most stable in terms of same execution times for different query samples.

For the comparison of numbers of individuals, we have created additional Individuals Databases with 10 times and 100 times as many individuals as in the initial version of the Individuals Database. Subsequently, we have carried out the same measurements for both versions as for the initial version. The measurement results again have shown that the Pellet reasoner with configuration 1 has performed best for the version with 10 times as many individuals. For the version with 100 times as many individuals, the Pellet reasoner has also performed best for the query-only execution time. Other reasoners could provide the same or slightly better results only for the execution time of the whole measurement step 2.

In addition to the previous comparison, we have also carried out measurements with yet other versions of the Individuals Database. In particular, we have created yet other Individuals Databases with 10 times and 100 times as many individuals as in the initial version. This time however, we have used another duplication approach, which has resulted in different interrelations between the individuals within the Individuals Databases. The results of the corresponding measurements and the comparison to the previous duplication approach again have shown that the Pellet reasoner with configuration 1 has performed best. However, it has also shown that for most query samples, the execution time has been slightly higher for the latter version compared to the previous version. For the KAON2 reasoner, on the other hand, there are huge runtime differences between these two versions. In particular, the KAON2 reasoner has needed up to 10 times as much query-only execution time for the latter version compared to the previous one. As explained in section 5.1.1, the Individuals Database includes the locations the user often visits and stays at, such as her home, the premises in which she works, and the environment in which she lives. Hence, the Individuals Database is user-specific, and could sometimes be similar to the one in the latter version, and sometimes to the one in the previous version. Consequently, we can not recommend the KAON2 reasoner for our user profile selection mechanism.

Finally, the comparison of different user profile selection approaches has shown that for the best performing reasoner, i.e. the Pellet reasoner with configuration 1, the query-only execution time for our ontology-based approach is about as fast as for other approaches. In particular, we have compared our ontology-based user profile selection mechanism with a classification-based user profile selection mechanism. As well, we have compared our approach with the fastest way to process a simple RDF document with the Jena2 library. In this approach, the user profile selection mechanism has received the user's current situation as an RDF document and has directly queried it without loading the user profile ontology, the context ontology and the Individuals Database.

The reverse conclusion for these comparison results is that the predominant part of the overall execution time concerns the preparation of the ontology model and the preparation of the reasoner. However, whereas the query-only execution time has to be considered and added up for every situational condition in the queried user profile, the preparation of the ontology model and the preparation of the reasoner has to be done only once for each user profile selection process. Consequently, the preparation of the ontology model and the reasoner only has to be done at the beginning of the user profile selection process, i.e. in case the user's situation changes and a new user profile selection process is started.

Besides the comparison of runtime performance, we have also compared the functionality that is supported by these three user profile selection approaches. Whereas our ontology-based user profile selection approach supports all operators introduced in section 4.3.1, the classification-based approach only supports a small subset of these operators, and the third approach explained above even supports less operators. Hence, the ontology-based approach provides the most expressiveness. Consequently, the ontology-based approach is the best approach for enabling expressive situational conditions for situation-dependent user preferences as introduced in section 3.1.1.

From our point of view, the Pellet reasoner with configuration 1 provides acceptable performance in realistic scenarios for many non-time-critical applications. With the initial Individuals Database, the measurement step 2 described in section 6.1.3, i.e. the preparation of the ontology model, the preparation of the reasoner and the first query, would be finished about 900 milliseconds after a situation change of a user has been reported to the user profile selection mechanism. Every subsequent query would in this case additionally need about 100 milliseconds, where the number of needed queries is at most the overall number of different situational conditions in the corresponding user profile. This result also shows, why it is important to cluster situation-dependent user data based on situational conditions, as we have proposed it with our user profile structure and ontology. The clustering can avoid redundant queries, and hence helps to reduce the overall execution time.

Even for the Individuals Databases with 10 times as many individuals as in the initial version, the preparation of the ontology model, the preparation of the reasoner and the result for the first query would be finished 2.2 seconds after a situation change of a user has been reported to the user profile selection mechanism. Every subsequent query would in this case additionally need about 200 milliseconds. Furthermore, other experiments have shown that the runtime performance of ontology reasoners can be improved when extending the main memory of the system and at the same time increasing the available memory for the Java virtual machine with the corresponding start argument.

7 Conclusion

In this closing chapter we first summarise the content and contribution of this thesis, which consists of the following:

- Design of a suitable user profile structure
- Analytic evaluation of search tasks to show the benefits of our user profile structure
- Design and implementation of a suitable user profile management system
- Modelling of a user profile ontology
- Modelling of a location ontology
- Design and implementation of a user profile selection mechanism
- Evaluation of our user profile selection mechanism in terms of runtime performance and supported functionality
- Experience report on using ontology reasoning capabilities

Second, the results of this doctoral thesis are evaluated against the requirements and goals stated in the introduction chapter of this thesis. Third, the implications of the suggested user profile management approach for service platforms for networks beyond 3G are discussed. Again, it should be mentioned that the service platform model referred to and described in this thesis is only a simplification of real-world service platforms. In reality, such service platforms are much more complex, including numerous layers and enablers, which are not addressed here.

Fourth, we discuss the results of our approach and the corresponding evaluation. Finally, the outlook addresses future work and issues that are related to this thesis, but have not been discussed in detail here.

7.1 Summary

As mentioned in the introduction chapter, the focus of this thesis is on the management and automatic selection of situation-dependent user preferences in context-aware service platforms for future mobile telecommunications systems. As the basic requirement, we have identified the research and development of a user profile management system that manages multiple user profile data required by different kinds of services. Furthermore, this user profile management system was to consider the use of different vocabularies for expressing user attributes. It was also intended to support automatic situation-dependent service personalisation without the need for user interaction. And, last but not least, it was to provide an easily understandable way for the normal non-technical user to specify situation-dependent user preferences. However, the main focus of this thesis has been on the automatic selection of matching situation-dependent user preferences combined with an easily understandable way to define expressive situation-dependent user preferences. In doing so, the user should not be faced with the complexity of the underlying user profile selection mechanism and technology.

To reach this goal, we first discussed design issues related to user profile structures for service platforms and introduced our user profile structure. This user profile structure enables the specification of application-specific user sub-profiles as required in service platforms to enable different application-specific user attribute vocabularies. Besides application-specific user sub-profiles, the user profile structure also enables the definition of situation-specific user preferences and situation-specific sub-profiles by adding situational conditions. Situational conditions are conditions that describe the situation, in which situation-dependent user preferences are relevant. These two properties are not fully covered with existing user profile structures and user profile ontologies such as GUP [36] [37], CC/PP [38], FOAF [41], vCard [42] or GUMO [44] [45] [46]. The situational conditions have been designed with the aim to be easily understandable and editable for the normal non-technical user on the one hand, but still being expressive enough to enable useful and detailed conditions on the other hand. For this purpose, an approach has been chosen, which is similar to a simple clause in the English language of the form <subject> <verb> <object>. In combination with a user profile editor, the definition of such situational conditions can be carried out in just selecting items from select lists. This means that for each of the attributes of a situational condition, the user does not have to fill out a blank text field, but can select from a pre-defined list.

Furthermore, we have also provided an analytic evaluation of runtime performance concerning two different search tasks for this user profile structure. The analytic evaluation has compared the required comparison steps for finding matching situation-dependent user data between different user profile structures. The results have shown that clustering user profile data into several situation-specific user sub-profiles, as is done in our user profile structuring approach, is beneficial with regard to the runtime performance for the targeted search tasks.

Subsequently, we discussed design issues for user profile management and presented our user profile management framework, which is a flexible modular framework that implements the specific requirements for the targeted environment. Existing research approaches and specifications such as GUP [37], GUC [58] [59] and others [61] [62] [63] [64] do not fully cover the envisioned functionality for service platforms in the telecommunications environment. For example, most of them do not include a mechanism for the automatic selection of the matching situation-dependent user preferences.

Our user profile management framework consists of three layers and several sub-modules, whereof several sub-modules are easily exchangeable. As a result, the user profile management framework is easily adaptable to different requirements as could arise in different service platform environments. A detailed activity diagram has been provided that depicts the inter-working of sub-modules and explains the steps carried out during user profile request by clients of the user profile management framework. In addition, also the interaction with a context management framework as part of the same service platform has been shown. Due to the integration into this context management framework the user profile management system can offer subscription/notification functionality beside the request/response communication pattern.

In the next chapter we addressed ontology modelling issues, showed the need to develop ontologies in service platforms and explained a potential approach for an ontology infrastructure for this environment. After this, we introduced our user profile ontology as a sub-ontology within such an ontology infrastructure. The specification of the user profile ontology is based on the user profile structure introduced before. Besides a description of ontology classes and properties, we also provided visualisations of the ontology, and a discussion on how the ontology can be extended with individual user attribute vocabularies on the one hand, and with individual context ontologies on the other hand.

We also presented our location ontology as an example of a context ontology for service platforms. However, the same modelling approach as shown in that section could also be used for other kinds of context ontologies, such as ontologies for user activities. We described how to design context ontologies in such a way that we can develop an advanced user profile selection mechanism that is based on ontology reasoning. The approach we followed is not the development of a user profile selection mechanism based on arbitrary context ontologies. Instead, both development steps have been linked together. This means that the development of the example location ontology is based on the goals we follow with our user profile selection mechanism. The location ontology has also been presented by means of class and property descriptions, as well as visualisations and descriptions of the class hierarchy.

In the subsequent chapter, we have addressed design and implementation of our User Profile Selection Module, which is part of our user profile management framework. The User Profile Selection Module carries out the selection of the matching user sub-profile for the user's current situation. This is done by applying ontology reasoning in the selection process. First, we explained the general functioning of the module by means of its sub-modules and processing steps. Afterwards, we depicted the actual selection algorithm by means of a detailed activity diagram. We have also depicted the details of several query processes. In particular, we created example user situations and example situational conditions as a starting point for the query process. Afterwards, the corresponding query was introduced and possible solutions for the inference step were shown that satisfy the corresponding query.

Finally, the evaluation of our user profile selection approach in terms of runtime performance and supported functionality was shown. For this purpose, the setup for our user profile selection mechanism and the corresponding runtime measurements were described, the characteristics of our user profile ontology, our context ontology and our database in terms of number of triples, number of classes, number of properties, and number of individuals were depicted, and also the different measurement steps, the measurement procedure and the measurement samples were introduced.

Subsequently, we depicted the actual measurement results for several runtime comparisons. In particular, we showed the results for the comparison of reasoner libraries, query samples, numbers of database entries, and user profile selection approaches. The results show that there are considerable differences in execution time between different reasoners. In summary, the Pellet reasoner library [30] has clearly outperformed all other reasoner libraries for most measurements. The comparison of user profile selection approaches has also shown that for the best performing reasoner, i.e. the Pellet reasoner, the query-only execution time for our ontology-based approach is about as fast as for other approaches that do not include ontology reasoning steps. It should be mentioned that the query-only execution time does not include the execution time needed for the preparation of the ontology model and the reasoner.

Beside the comparison of runtime performance, we also compared the functionality that is supported by different user profile selection approaches. We clearly showed that the ontology-based approach provides the most expressiveness. Hence, the ontology-based approach has shown to be the best approach for enabling expressive situational conditions for situation-dependent user preferences.

7.2 Evaluation

The evaluation in this section evaluates the results of this thesis against the requirements and goals stated in section 1.3. For a better overview, first the goal stated in that section is repeated, and afterwards evaluated:

Goal 1:

- Design and implementation of a user profile management system that is capable of managing the following kinds of user data
    a. User data required by platform services such as accounting and billing services
    b. User data required by end user services such as innovative messaging services and location-based services
    c. Service-specific user data that is only required by specific services such as ring tone preferences for telephony services
    d. Situation-dependent user data such as notification preferences for incoming new messages that are related to different situations

Evaluation of goal 1:
As described in the last subsection, the designed and implemented user profile management system, in combination with the developed user profile structure, is flexible enough to fulfil all the different aspects of goal 1. In particular, the user profile structure enables the specification of service-specific sub-profiles within an overall user profile. These sub-profiles can serve different kinds or services, such as platform services and end user services. In addition, the different sub-profiles can be filled with arbitrary service-specific user attributes. Last but not least, these sub-profiles can optionally be enhanced with situational conditions, which enable the specification of single situation-dependent user preferences or clusters of situation-dependent user data.

Goal 2:

- Design and implementation of a user profile management system that is capable of managing user data that adheres to different and extensible vocabularies for expressing user attributes
    a. It should be possible to manage user data that adheres to different user attribute vocabularies
    b. It should be possible to manage user data that adheres to extensions of user attribute vocabularies

Evaluation of goal 2:
The user profile management system is designed and implemented in a way that arbitrary user information can be managed. That is, there is no restriction on a specific set of user attributes, user attribute vocabulary or amount of user data. The only limitation is in the requirement on how user data is encoded. In particular, the actual user data that is managed by the user profile management system has to adhere to the RDF specification [17] and can reference arbitrary user attribute vocabularies. It can also reference definitions from different user attribute vocabularies or arbitrary extensions of those.

The actual user profile structure, enabling service and situation-dependent user sub-profiles is managed independently from the user attribute vocabularies. In particular, the user profile structure is an integral part of the user profile management system used in a service-independent way, whereas the user attribute vocabularies can be service specific. The situational conditions, however, are also part of this user profile structure. This is needed to provide a service-independent user profile selection module that needs to process these situational conditions.

In summary, user data can be expressed in a service-specific way adhering to arbitrary user attribute vocabularies. Only the situational conditions of situation-dependent user data need to be expressed in a predefined service-independent way. This actually makes sense, as not the services, but the user profile management system evaluates these situational conditions. Hence, all aspects of goal 2 are met with this solution.

Goal 3:

- Design and implementation of a user profile management system that supports automatic service personalisation
    a. Design and implementation of a user profile selection module as sub-module of the user profile management system that enables automatic selection of matching situation-dependent user preferences
    b. Design and implementation of a module to receive and process context parameters as input to the user profile selection module

Evaluation of goal 3:
The user profile management system is designed and implemented in a modular way consisting of several exchangeable sub-modules, whereof the user profile selection module and the context client module are two of them. The context client module interacts with external context providers to request and collect context information about the user's environment and to pass it on to the user profile selection module for further processing.

These context descriptions are also encoded in RDF, as is the user profile data, and adhere to context vocabularies used throughout the whole service platform infrastructure. The user profile selection module supports situation-dependent service personalisation in finding the matching situation-dependent user data to be applied for service personalisation. This selection functionality is implemented using ontology reasoning capabilities that enable sophisticated evaluations as is described in various samples in this thesis. This thesis also presents a runtime evaluation concerning different ontology reasoning libraries and input parameters, which shows the current state of art, state of usability, advantages and disadvantages concerning the young discipline of ontology reasoning. Besides meeting all aspects of goal 3, the use and evaluation of ontology reasoning capabilities shows the advantage of clustering situation-dependent user data for runtime improvements as is done with the developed user profile structure of goal 1.

Goal 4:

- Design and implementation of a user profile management system that supports an easily understandable way for the normal user to manage her situation-dependent user data
    a. It should be possible for the user to easily specify and edit situation-dependent user preferences
    b. It should be possible for the user to control user data and personalisation features

Evaluation of goal 4:
This thesis does not include solutions for user profile editors. However, the user profile management system, in combination with the developed user profile structure provides a base for an easily understandable way for the normal user to manage her situation-dependent user data. The situational conditions are similar to a simple clause in English language of the form <subject> <verb> <object>. This can be visualised to a user in an easy way, using select lists to fill these fields in order to create and edit such situational conditions.

Furthermore, as already mentioned, the user profile management system can also be used by platform services for managing user data. This means that also user preferences about general personalisation features within the service platform can be added to the user profile management system and hence provided to the user for controlling purposes. Hence, the aspects of goal 4 are not directly implemented by the user profile management system, but are supported indirectly. It is now up to other functionality, e.g. user profile editors and general service platform concepts to implement these features.

7.3 Implications and Suggestions

The suggested approach on a user profile management system, user profile structure and related issues that are designed and implemented in this thesis leads to several implications and suggestions. As our focus was on service platforms for networks beyond 3G, the following implications and suggestions are directed to designers of services and service platforms of this domain. The implications and suggestions are as follows:

Implication 1: Lifecycle management of the user profile management service
Usual end-user services in the addressed type of service platform are subject to a service lifecycle management, usually including at least the steps *install*, *deploy*, *start*, *stop*, *undeploy*, *uninstall*, clearly mentioning that these steps may be named differently or may be extended or substituted by further or different steps. However, the user profile management as suggested in this thesis must be considered as specific service that is essential for providing personalised end-user services. That is, this service must be available, i.e. started, before end-user services can be consumed by end-users to support intelligent service behaviour such as customisation to the user's situation. Consequently, it also cannot be stopped during operation of the service platform, but the lifecycle management could e.g. provide a means to upgrade it while running.

Implication 2: Lifecycle management of end-user services
End-user services cannot be supported with intelligent service behaviour provided by the user profile management service before service-related user data and optional situational conditions have been added to the user profile management service. Hence, it should be thought about adding further service lifecycle management states. For instance, you could add a state *customisation support* that can be reached from the *start* state in setting up the service-specific user profile. This could happen via a user profile editor provided by the end-user service.

Implication 3: Design of context ontologies used on service platform side
In order for the user profile management, in particular for the user profile selection module, to provide intelligent service behaviour, some design requirements are given for the representation of context parameters on service platform side. As shown in this thesis, we suggest an ontology reasoning based approach for the selection of matching situation-dependent user preferences. This ontology reasoning based approach assumes the context to be represented and exchanged as RDF documents. However, in order to ensure computational completeness (all entailments are guaranteed to be computed) and decidability (all computations will finish in finite time) in the reasoning step, the context schemas need to adhere to the OWL species OWL-DL (Description Language), see [20]. Hence, service platform designers, in particular designers of context representations used within the service platform should think about modelling context accordingly and supporting the mentioned representation.

Implication 4: Design of user profiles for designers of end-user services
Designers of end-user services do not need to care about own user profile management. However, in order to be supported with intelligent service behaviour by the service-side user profile management service, they need to process user data in a specific manner, as already described in the previous subsection. In particular, they need to encode user data as an RDF document. Beforehand, they first need to decide on one or several user attribute vocabularies. It is recommended to use user attribute vocabularies that are already available on service platform side, as this will increase the reuse and sharing of user attributes with other services served by the service platform. The advantage of reuse and sharing of user attributes for the end-user means that she may not have to define user attributes once again that are already available on platform side, e.g. because these user attributes have already been added for other end-user services.

Implication 5: Design of end-user services
Designers of end-user services should be aware of the intelligent functions the platform-side user profile management provides. In particular, end-user service designers do not need to design and implement own context gathering and processing functions, and they do not need to design and implement a selection or reasoning function for customising the end-user service to the user's current situation. Both these function are usually considered as complex functions because of the requirement of sophisticated reasoning means. This means that the end-user service requires less complexity, less development time and finally may require less development costs. However, end-user service designers should check, which types of context are supported and processed on service platform side and which of them are taken into account by the user profile management service. There may be specific kinds of context information that is not supported by service platforms for two reasons. Firstly, some context parameters could be very specific, e.g. context information in the domain of health applications, which may be very specific for some end-user services. Secondly, there may be end-user services, e.g. also in the area of health applications, which require time critical behaviour. In this case, the end-user service designer should be aware of the runtime evaluations shown in this thesis for the proposed ontology-based reasoning approach. As already mentioned, the runtime performance may be sufficient for many non-time critical application, but may be problematic for some time-critical application such as in the area of health applications.

Although we focused on service platforms for networks beyond 3G, we believe that many parts of the depicted approach in this thesis could also be used for service platforms in other domains. For instance, we consider the user profile structure flexible enough to serve for similar purposes in other kinds of service platforms or multi-application environments, in which multiple services, service personalisation, user attribute vocabularies and sharing and reuse of user data is desired.

7.4 Discussion

The research and development steps carried out in this thesis have been built on top of each other. However, the user profile management framework has been designed with flexibility in mind. Besides several sub-modules of the user profile management framework, also the user profile structure and the corresponding user profile ontology could be substituted. In particular, it might not always be desired to cluster user data and user preferences into situation-specific user sub-profiles in the same way as we have done it. For this purpose, it is also possible to substitute the user profile structure and the user profile ontology introduced here with another user profile structure and user profile ontology. However, the presented user profile selection mechanism can still be used for other user profile structures and user profile ontologies as long as the situational conditions of situation-dependent user preferences have the same structure as the ones we have introduced.

We would also like to add that the modelling of the location ontology and the corresponding situational conditions should be seen as an example of how a user profile selection mechanism can be realised based on ontology reasoning. Hence, the presented solution is not the only way (and may not be the best way) how to model such an ontology-based user profile selection mechanism. However, the runtime performance carried out could still be representative for slightly different location ontologies as our evaluation has included often recurring ontology reasoning tasks such as reasoning over symmetric and transitive properties, and reasoning over class and property hierarchies.

From our point of view, the Pellet reasoner [30] provides acceptable performance in realistic scenarios for many non-time-critical applications. By using the introduced ontologies, the whole query process needs about 900 milliseconds plus about 100 milliseconds for each additional query, where the number of needed queries is at most the overall number of different situational conditions in the corresponding user profile. The mentioned query process includes the preparation of the ontology model, the preparation of the reasoner and the execution of the first query for one of the situational conditions. The fact that one query needs about 100 milliseconds shows, why it is important to cluster situation-dependent user data based on situational conditions as we have proposed it with our user profile structure and ontology. The clustering can avoid redundant queries, and hence helps to reduce the overall execution time.

Using about 10 times as many individuals, e.g. rooms and buildings, in our database, the whole query process needs about 2.2 seconds plus about 200 milliseconds for each additional query. Some additional experiments have shown that the runtime performance of ontology reasoners can be improved when extending the main memory of the system and at the same time increasing the available memory for the Java virtual machine with the corresponding start argument.

The results also show that there are considerable differences in execution time and in behaviour between different reasoners. This is mainly caused by the fact that ontology reasoning is a quite new discipline, standards for ontology languages and query languages are still in progress, and different types of algorithms for ontology reasoning are still being researched and developed. At the same time, unfortunately, the use and configuration of reasoning libraries is not always well documented.

Anyway, it should be stressed that the results of the runtime evaluation in this thesis are biased due to several reasons: choice of hardware and software environment as described in section 6.1.1, implementation approach as described in section 3.2.1 and 5.1.1, design of ontologies as described in chapter 4 and dimension of database as described in section 6.1.2. The outcome of this is that there may me a significant opportunity for improving the performance compared to the results described. For instance, the thesis showed that there are significant runtime differences for different dimensions of databases, there may be significant runtime improvements with faster and dedicated hardware and there may be significant improvements in ontology reasoning libraries in future.

The results of this thesis can be used in two ways. Firstly, they can be used for the targeted environment. The user profile management system, the user profile structure and ontology, as well as the user profile selection functionality can be used for the objectives of service platforms in telecommunications environments. In particular, they can be used to enable sophisticated personalised services that adapt to the user's current situation.

Secondly, the results of this thesis can be used by various application developers that aim to use Semantic Web technologies and ontology reasoning capabilities in practical applications. This is not necessarily limited to developers that work in the targeted service platform environment, but also in other areas. They can use the findings of this thesis as instruction on how to model ontologies, how to use ontology reasoning capabilities, and to understand the advantages and disadvantages of using ontology reasoning capabilities. Furthermore, they can also use it as experiences report. In doing so, they can estimate the system properties in the sense of execution time with regard to used software environment, hardware environment, size of database and other parameters of their systems. This can be done based on the measurement results carried out in this thesis. With the resulting estimation, they can then plan their system accordingly beforehand.

7.5 Outlook

There are several issues related to our user profile management framework that have not been covered in detail by this thesis. Firstly, we have not addressed the issue of user modeling, i.e. we have not learned user models. In particular, we were not concerned about how user attributes are added to a user profile. However, the user profile structure is not limited to a certain user attribute vocabulary. Individual user attribute vocabularies could be used that are designed to express learned user models as well as explicitly specified user attributes by users of service platforms. Hence, the challenge for integrating learned user models into our user profile management framework is the creation of the corresponding user attribute vocabularies. In addition, user profile editors are needed for the visualisation of user attributes to the user and for making user attributes editable by the user.

Secondly, privacy and security issues related to personal data are beyond the scope of this thesis. However, within a service platform in telecommunications environments, privacy is not only an issue for the user profile management framework. Instead, also user context such as user location and user activity, user behaviour and other user-related data collected in a service platform have to be addressed. Hence, a service platform spanning privacy and security architecture is needed that also covers sensitive user profile data.

For our user profile selection mechanism, we have assumed that the received user situation is accurate. That is, we have not dealt with probabilities. However, as low-level sensor data is usually not 100 percent accurate, the resulting high-level user situation can only be inferred with a certain probability. Hence, inaccuracy of information about user situations should be taken into consideration. This could be done by extending our user profile selection mechanism to a recommendation mechanism.

We have also assumed that a model of the user’s world is available for the user profile selection mechanism that includes the locations the user often visits and stays at, such as her home, the premises in which she works, and the environment in which she lives. Such a model could be created by a system administrator or another expert at the time when the user subscribes to the service platform. However, as this appears to be a quite time-consuming and hence cost-intensive task, it should be aimed at supporting and automating this task. For instance, this could be supported by means of learning mechanisms that learn the locations the user often visits and stays at, and adds these locations to the user-specific location database.

Furthermore, we have assumed that there are no situation-specific user sub-profiles with conflicting or overlapping situational conditions. However, in order to achieve this, situation-specific user sub-profiles and the corresponding situational conditions have to be checked during creation time. For this purpose, an algorithm is needed to detect conflicting and overlapping situation-specific user sub-profiles.

As mentioned in the last section, standards for ontology languages and query languages are still in progress, and algorithms and reasoning libraries for ontology reasoning are still being researched and developed. Hence, we expect improvements in reasoning algorithms in the coming years, which make ontology reasoners faster and more stable, and hence are also beneficial for our user profile selection mechanism in terms of execution time.

Finally, the success and acceptability of personalised services, and hence of our user profile selection mechanism, is decided by the user of the service platform. In order to investigate its success and acceptability, and also to identify possible problems and develop improvements, user evaluations have to be carried out in the future.

Appendix A: SPICE Service Platform

In this appendix, we show details of the service platform developed in the SPICE [4] [5] [6] project. The details are provided on the level of functional blocks; see Figure 63, which is taken from [97]. Most of these functional blocks are composed of many subcomponents working together, which are not shown here. For even more details and the descriptions of the functional blocks, please see the SPICE project deliverable on the final reference architecture [97] of the SPICE service platform.

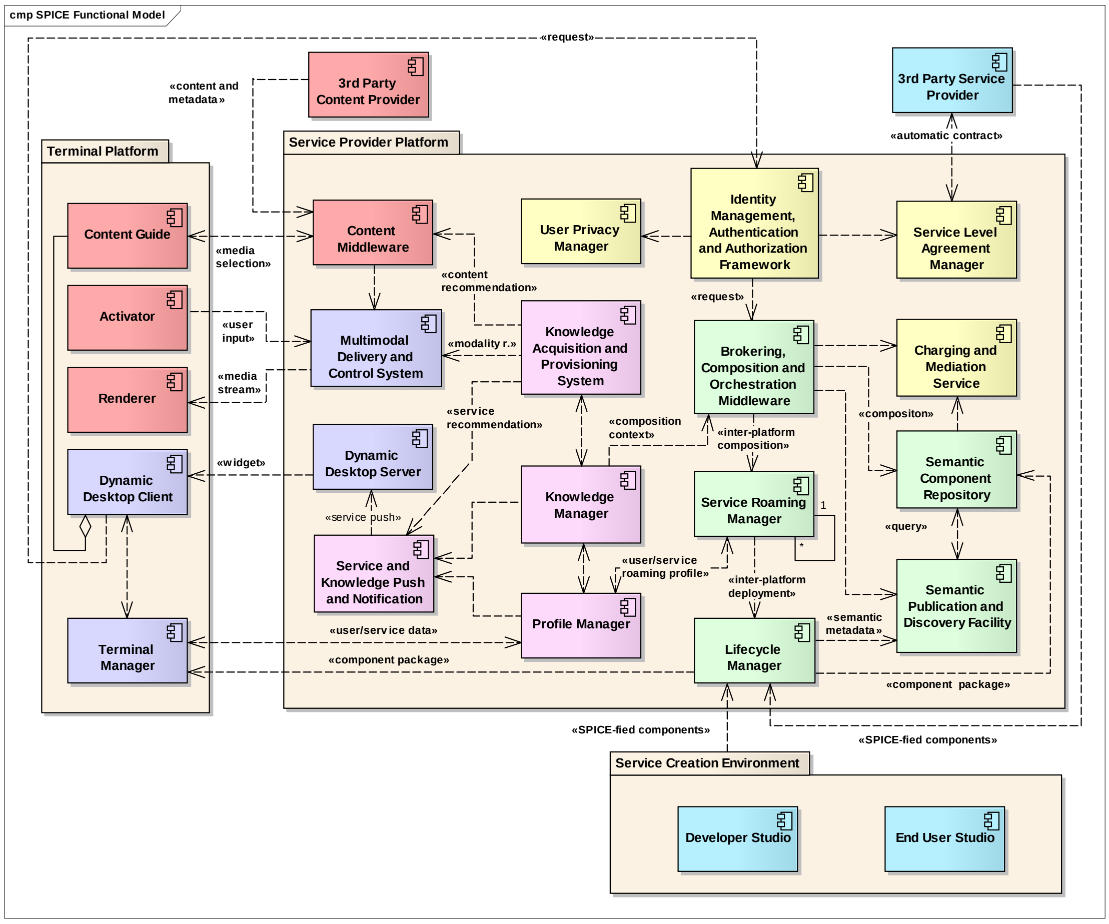
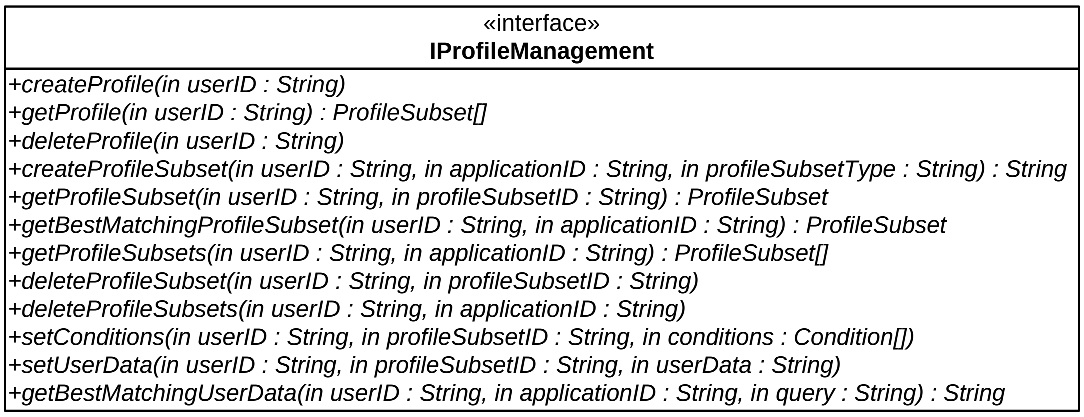


Figure 63: SPICE Functional Blocks (Source: [97])

Appendix B: User Profile Management Interface

Figure 64 shows the most important methods of the user profile management interface. These methods are explained below. The user profile management framework is implemented in the Java programming language from Sun Microsystems. However, for the purpose of interoperability, the implementation of the user profile management framework also provides a Web service [98] [99] interface. Afterwards, documentation for users of the user profile management framework and developers of single modules is added.



```
«interface»
IProfileManagement
+createProfile(in userID : String)
+getProfile(in userID : String) : ProfileSubset[]
+deleteProfile(in userID : String)
+createProfileSubset(in userID : String, in applicationID : String, in profileSubsetType : String) : String
+getProfileSubset(in userID : String, in profileSubsetID : String) : ProfileSubset
+getBestMatchingProfileSubset(in userID : String, in applicationID : String) : ProfileSubset
+getProfileSubsets(in userID : String, in applicationID : String) : ProfileSubset[]
+deleteProfileSubset(in userID : String, in profileSubsetID : String)
+deleteProfileSubsets(in userID : String, in applicationID : String)
+setConditions(in userID : String, in profileSubsetID : String, in conditions : Condition[])
+setUserData(in userID : String, in profileSubsetID : String, in userData : String)
+getBestMatchingUserData(in userID : String, in applicationID : String, in query : String) : String
```

Figure 64: Extract of Profile Management Interface

CreateProfile(in userID: String)
This method is used to create a user profile. The userID is the identifier of the user within the service platform.

GetProfile(in userID: String): ProfileSubset[]
This method is used to request the whole user profile of a particular user, i.e. the list of all profile subsets of the corresponding user.

DeleteProfile(in userID: String)
This method is used to delete the whole user profile. In doing so, all related profile subsets and user data are also deleted.

CreateProfileSubset(in userID: String, in applicationID: String, in profileSubsetType: String): String
This method is used to create an application-specific profile subset for a specific user. The profile subset type can have the values *default* for creating a Default Profile Subset or *conditional* for creating a Conditional Profile Subset. A user profile can only include one Default Profile Subset related to each application. Furthermore, Conditional Profile Subsets can only be created after a Default Profile Subset for the related application has been created. The return value includes the generated profile subset identifier, which is unique within a user profile.

GetProfileSubset(in userID: String, in profileSubsetID: String): ProfileSubset
This method is used to request a certain profile subset of a particular user profile. For this purpose, the *profileSubsteID* is unique within a user profile.

GetBestMatchingProfileSubset(in userID: String, in applicationID: String): ProfileSubset
This method is used to request the best matching profile subset of a particular user profile concerning a certain application. That is, the profile subset that best matches the user's current situation is returned. This method is explained in more detail in section 3.2.1.

GetProfileSubsets(in userID: String, in applicationID: String): ProfileSubset[]
This method is used to request all profile subsets of a particular user profile that are related to a certain application.

DeleteProfileSubset(in userID: String, in profileSubsetID: String)
This method is used to delete a certain profile subset of a particular user profile.

DeleteProfileSubsets(in userID: String, in applicationID: String)
This method is used to delete all profile subsets of a particular user profile that are related to a certain application.

SetConditions(in userID: String, in profileSubsetID: String, in conditions: Condition[])
This method is used to set the conditions of a Conditional Profile Subset. If there are several conditions, the conditions are interpreted as conjunct to each other.

SetUserData(in userID: String, in profileSubsetID: String, in userData: String)
This method is used to set the user data of a profile subset. User data is encoded as an RDF/XML serialisation of an RDF graph as explained in section 3.1.1 and shown in Figure 12.

GetBestMatchingUserData(in userID: String, in applicationID: String, in query: String): String
This method is used to request the best matching user data of a particular user profile concerning a certain application. That is, the user data that best matches the user's current situation is returned. In comparison to the *getBestMatchingProfileSubset* method, this method does not return the best matching profile subset as a whole. Instead, only parts thereof are returned depending on the *query* parameter. The query is encoded as a SPARQL query as explained in more detail in section 3.2.1 and shown in Figure 22. The result is encoded in XML as shown in Figure 23.

In the following, we provide a small documentation for users of the user profile management framework on how user profiles are created, how profile subsets are created and how user data is requested.

```
//Use the User Profile Management interface
IProfileManagement pm= new CProfileManagement();

//Define a userID for user Tom
String userID= "tom";

//Create a user profile for user Tom in case it does not exist yet. In case a user
//profile with the same user ID already exists, an exception is returned.
pm.createProfile(userID);

//Define an application ID for the (group of) application(s) for
//which to create a Profile Subset
String appID= "myApp";

//Define the type of Profile Subset to be created.
//If Default Profile Subset: set “default”
//If Conditional Profile Subset: set “conditional”
String profileSubsetType= "default";

//Create a Default Profile Subset for user Tom
//The returned profile subset ID is unique within this user profile
String profileSubsetID= pm.createProfileSubset(userID, appID, profileSubsetType);

//Define and set default user data for user Tom
String userData= …//Tom’s default user data as RDF/XML serialisation
pm.setUserData(userID, profileSubsetID, userData);
```

Figure 65: Creation of User Profile and Default Profile Subset

Figure 65 shows the creation of a user profile, the creation of a Default Profile Subset and the set-up of a Default Profile Subset.

```
//Create array of situational conditions
Condition[] conditions= new Condition[1];

//Set attributes of a situational condition
String entityID= “tom”;
String contextType= “location”;
String operator= “equals”;
String contextValue= “Room-25”;

//Instantiate the situational condition
conditions[0] = new Condition(entityID, contextType, operator, contextValue);

//Add situational conditions to Conditional Profile Subset
pm.setConditions(userID, profileSubsetID, conditions);

//Define and set conditional user data for user Tom
String userData= …//Tom’s conditional user data as RDF/XML serialisation
pm.setUserData(userID, profileSubsetID, userData);
```

Figure 66: Set-up of Conditional Profile Subset

Figure 66 shows the set-up of a Conditional Profile Subset. Details on conditions are explained in chapter 5.

```
//Use the User Profile Management interface
IProfileManagement pm= new CProfileManagement();

//Define the user ID of the related user
String userID= “tom”;

//Define the application ID of the querying application
String appID= “myApp”;

//Define the optional SPARQL query
//If not needed: set null
String query= …//SPARQL query

//Request best matching user data
String userData= pm.getBestMatchingUserData(userID, appID, query);
```

Figure 67: Request of Best Matching User Data

Figure 67 shows the request of best matching user data, i.e. the user data that best matches the current user’s situation.

Developers of single modules for the user profile management framework can easily exchange the User Profile Selection Module, the Context Client Module, the User Profile Sharing Module and the Database Management System. The mechanism is the same for all four modules. First, the corresponding interface has to be implemented. These interfaces are part of the User Profile Management Core. Second, the name of the implementation class of the corresponding module has to be set as value of the corresponding key in a properties file. This properties file is also part of the User Profile Management Core.

Appendix C: Specification of the User Profile Ontology

This appendix includes the specification of our User Profile Ontology introduced in section 4.2. The specification is done with OWL and is represented in Notation 3 (N3) format [100]. N3 is a serialisation format for RDF and OWL respectively:

```
@prefix xsd:      <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs:     <http://www.w3.org/2000/01/rdf-schema#> .
@prefix daml:     <http://www.daml.org/2001/03/daml+oil#> .
@prefix rdf:      <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix owl:      <http://www.w3.org/2002/07/owl#> .
@prefix :         <http://ws.comtec.e-technik.uni-kassel.de/upos.owl#> .

<http://ws.comtec.e-technik.uni-kassel.de/upos.owl>
    a       owl:Ontology ;
    rdfs:comment "The User Profile Ontology with Situation-Dependent Preferences Support (UPOS) defines concepts and properties for a user profile that supports situation-dependent sub-profiles. The ontology can be extended with various user attribute vocabularies such as vCard and FOAF. Copyright 2007 Michael Sutterer, Kassel, Germany."^^xsd:string .

:User
    a       owl:Class ;
    rdfs:comment "A user of the service platform."^^xsd:string .

:hasContext
    a       owl:ObjectProperty ;
    rdfs:comment "A user is in a certain context."^^xsd:string ;
    rdfs:domain :User ;
    rdfs:range :Context .

:Context
    a       owl:Class ;
    rdfs:comment "A context, e.g. a user activity or a user location."^^xsd:string .

:hasLocation
    a       owl:ObjectProperty ;
    rdfs:comment "A user has a location."^^xsd:string ;
    rdfs:domain :User ;
    rdfs:range :Location ;
    rdfs:subPropertyOf :hasContext .

:Location
    a       owl:Class ;
    rdfs:comment "A user location."^^xsd:string ;
    rdfs:subClassOf :Context .

:hasActivity
    a       owl:ObjectProperty ;
    rdfs:comment "A user is engaged in an activity."^^xsd:string ;
    rdfs:domain :User ;
```

```
     rdfs:range :Activity ;
     rdfs:subPropertyOf :hasContext .

:Activity
     a       owl:Class ;
     rdfs:comment "A user activity."^^xsd:string ;
     rdfs:subClassOf :Context .

:hasProfile
     a       owl:ObjectProperty , owl:FunctionalProperty ;
     rdfs:comment "A user has a profile."^^xsd:string ;
     rdfs:domain :User ;
     rdfs:range :Profile .

:Profile
     a       owl:Class ;
     rdfs:comment "A user profile."^^xsd:string .

:hasProfileSubset
     a       owl:ObjectProperty ;
     rdfs:comment "A user profile has a profile subset."^^xsd:string ;
     rdfs:domain :Profile ;
     rdfs:range :ProfileSubset .

:ProfileSubset
     a       owl:Class ;
     rdfs:comment "A service-specific and optionally situation-specific subset of a user
profile."^^xsd:string ;
     rdfs:subClassOf owl:Thing ;
     rdfs:subClassOf
             [ a       owl:Restriction ;
               owl:cardinality "1"^^xsd:int ;
               owl:onProperty :isSpecificTo
             ] ;
     rdfs:subClassOf
             [ a       owl:Restriction ;
               owl:cardinality "1"^^xsd:int ;
               owl:onProperty :hasName
             ] .

:hasDefaultProfileSubset
     a       owl:ObjectProperty ;
     rdfs:comment "A user profile has a default profile subset. Related to each service, only
one default profile subset is allowed."^^xsd:string ;
     rdfs:domain :Profile ;
     rdfs:range :DefaultProfileSubset ;
     rdfs:subPropertyOf :hasProfileSubset .

:DefaultProfileSubset
     a       owl:Class ;
```

```
    rdfs:comment "A service-specific subset of a user profile that includes a default user
model."^^xsd:string ;
    rdfs:subClassOf :ProfileSubset .

:hasConditionalProfileSubset
    a       owl:ObjectProperty ;
    rdfs:comment "A user profile has a conditional profile subset."^^xsd:string ;
    rdfs:domain :Profile ;
    rdfs:range :ConditionalProfileSubset ;
    rdfs:subPropertyOf :hasProfileSubset .

:ConditionalProfileSubset
    a       owl:Class ;
    rdfs:comment "A service and situation-specific subset of a user profile that includes a user
model for a specific situation."^^xsd:string ;
    rdfs:subClassOf :ProfileSubset ;
    rdfs:subClassOf
            [ a       owl:Restriction ;
              owl:minCardinality "1"^^xsd:int ;
              owl:onProperty :hasCondition
            ] .

:hasName
    a       owl:DatatypeProperty , owl:FunctionalProperty ;
    rdfs:comment "A profile subset has a name. This name has to be unique within a user
profile."^^xsd:string ;
    rdfs:domain :ProfileSubset ;
    rdfs:range xsd:string .

:hasDescription
    a       owl:DatatypeProperty , owl:FunctionalProperty ;
    rdfs:comment "A profile subset has an optional description."^^xsd:string ;
    rdfs:domain :ProfileSubset ;
    rdfs:range xsd:string .

:isSpecificTo
    a       owl:ObjectProperty , owl:FunctionalProperty ;
    rdfs:comment "A profile subset is specific to a service."^^xsd:string ;
    rdfs:domain :ProfileSubset ;
    rdfs:range :Service .

:Service
    a       owl:Class ;
    rdfs:comment "A service associated with the service platform."^^xsd:string .

:hasUserModel
    a       owl:ObjectProperty , owl:FunctionalProperty ;
    rdfs:comment "A profile subset has a user model."^^xsd:string ;
    rdfs:domain :ProfileSubset ;
    rdfs:range :UserModel .
```

```
:UserModel
      a       owl:Class ;
      rdfs:comment "A user model."^^xsd:string .

:hasCondition
      a       owl:ObjectProperty ;
      rdfs:comment "A conditional profile subset has a condition."^^xsd:string ;
      rdfs:domain :ConditionalProfileSubset ;
      rdfs:range :Condition .

:Condition
      a       owl:Class ;
      rdfs:comment "A condition specifies the situation, in which a conditional profile subset is
valid, e.g. the home or the office location."^^xsd:string ;
      rdfs:subClassOf owl:Thing ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:cardinality "1"^^xsd:int ;
                owl:onProperty :hasEntity
              ] ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:cardinality "1"^^xsd:int ;
                owl:onProperty :hasContextValue
              ] ;
      rdfs:subClassOf
              [ a       owl:Restriction ;
                owl:cardinality "1"^^xsd:int ;
                owl:onProperty :hasOperator
              ] .

:hasEntity
      a       owl:ObjectProperty , owl:FunctionalProperty ;
      rdfs:comment "A condition is linked to an entity, e.g. a user or a room."^^xsd:string ;
      rdfs:domain :Condition ;
      rdfs:range :User .

:hasOperator
      a       owl:DatatypeProperty , owl:FunctionalProperty ;
      rdfs:comment "A condition has an operator, e.g. equal, notEqual, greaterThan,
greaterThanOrEqual, lessThan or lessThanOrEqual."^^xsd:string ;
      rdfs:domain :Condition ;
      rdfs:range xsd:string .

:hasContextValue
      a       owl:ObjectProperty , owl:FunctionalProperty ;
      rdfs:comment "A condition has a context value, e.g. a certain user location."^^xsd:string ;
      rdfs:domain :Condition ;
      rdfs:range :Context .
```

Appendix D: Additional Property Specification

This appendix includes the alternative specification of the hasContextValue property of the user profile ontology. The specification is done with OWL and is represented in Notation 3 (N3) format [100]. N3 is a serialisation format for RDF and OWL respectively. The first of the following specifications is a repetition of the initial specification in Appendix C: Specification of the User Profile Ontology, whereas the second specification is the alternative specification to meet the requirements discussed in section 4.3.1. The difference between the hasContextValue property in the initial and the alternative specification is the values allowed in the range of the property. Whereas the initial specification only allows individuals of type Context, e.g. the room MeetingRoom-1, the alternative specification allows either individuals of type Context or OWL classes, e.g. the class MeetingRoom.

```
:hasContextValue
    a       owl:ObjectProperty , owl:FunctionalProperty ;
    rdfs:comment "A condition has a context value, e.g. a certain user location."^^xsd:string ;
    rdfs:domain :Condition ;
    rdfs:range :Context .

:hasContextValue
    a       owl:ObjectProperty , owl:FunctionalProperty ;
    rdfs:comment "A condition has a context value, e.g. a certain user location."^^xsd:string ;
    rdfs:domain :Condition ;
    rdfs:range
            [ a       owl:Class ;
              owl:unionOf (:Context owl:Class)
            ] .
```

Appendix E: Specification of the Location Ontology

This appendix includes the specification of our Location Ontology introduced in section 4.3. The Location Ontology also contains some few concepts for user activities. The specification is done with OWL and is represented in Notation 3 (N3) format [100]. N3 is a serialisation format for RDF and OWL respectively:

```
@prefix xsd:     <http://www.w3.org/2001/XMLSchema#> .
@prefix rdfs:    <http://www.w3.org/2000/01/rdf-schema#> .
@prefix daml:    <http://www.daml.org/2001/03/daml+oil#> .
@prefix rdf:     <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix owl:     <http://www.w3.org/2002/07/owl#> .
@prefix :        <http://ws.comtec.e-technik.uni-kassel.de/context.owl#> .

<http://ws.comtec.e-technik.uni-kassel.de/context.owl>
    a       owl:Ontology ;
    rdfs:comment """The Context Ontology defines concepts and properties to describe the situation of a user, in particular the location, activity, time of day and weekday. Copyright 2007 Michael Sutterer, Kassel, Germany."""^^xsd:string ;
    owl:imports <http://ws.comtec.e-technik.uni-kassel.de/upos.owl> .

:isPartOf
    a       owl:ObjectProperty , owl:TransitiveProperty ;
    rdfs:domain <http://ws.comtec.e-technik.uni-kassel.de/upos.owl#Location> ;
    rdfs:range <http://ws.comtec.e-technik.uni-kassel.de/upos.owl#Location> ;
    owl:inverseOf :hasPart .

:hasPart
    a       owl:ObjectProperty , owl:TransitiveProperty ;
    rdfs:domain <http://ws.comtec.e-technik.uni-kassel.de/upos.owl#Location> ;
    rdfs:range <http://ws.comtec.e-technik.uni-kassel.de/upos.owl#Location> ;
    owl:inverseOf :isPartOf .

:isDirectPartOf
    a       owl:ObjectProperty ;
    rdfs:domain <http://ws.comtec.e-technik.uni-kassel.de/upos.owl#Location> ;
    rdfs:subPropertyOf :isPartOf ;
    owl:inverseOf :hasDirectPart .

:hasDirectPart
    a       owl:ObjectProperty ;
    rdfs:range <http://ws.comtec.e-technik.uni-kassel.de/upos.owl#Location> ;
    rdfs:subPropertyOf :hasPart ;
    owl:inverseOf :isDirectPartOf .

:isConnectedTo
    a       owl:ObjectProperty , owl:TransitiveProperty , owl:SymmetricProperty ;
    rdfs:domain
          [ a       owl:Class ;
            owl:unionOf (:Room_Business :Room_Private)
```

```
        ] ;
    rdfs:range
        [ a      owl:Class ;
          owl:unionOf (:Room_Business :Room_Private)
        ] ;
    owl:inverseOf :isConnectedTo .

:isDirectlyConnectedTo
    a      owl:ObjectProperty , owl:SymmetricProperty ;
    rdfs:domain
        [ a      owl:Class ;
          owl:unionOf (:Room_Business :Room_Private)
        ] ;
    rdfs:range
        [ a      owl:Class ;
          owl:unionOf (:Room_Business :Room_Private)
        ] ;
    rdfs:subPropertyOf :isConnectedTo .

:hasTimeOfDay
    a      owl:ObjectProperty ;
    rdfs:domain <http://ws.comtec.e-technik.uni-kassel.de/upos.owl#User> ;
    rdfs:range :TimeOfDay ;
    rdfs:subPropertyOf <http://ws.comtec.e-technik.uni-kassel.de/upos.owl#hasContext> .

:TimeOfDay
    a      owl:Class ;
    rdfs:subClassOf <http://ws.comtec.e-technik.uni-kassel.de/upos.owl#Context> .

:hasWeekDay
    a      owl:ObjectProperty ;
    rdfs:range :WeekDay ;
    rdfs:subPropertyOf <http://ws.comtec.e-technik.uni-kassel.de/upos.owl#hasContext> .

:WeekDay
    a      owl:Class ;
    rdfs:subClassOf <http://ws.comtec.e-technik.uni-kassel.de/upos.owl#Context> .

:PrivatePlace
    a      owl:Class ;
    rdfs:subClassOf <http://ws.comtec.e-technik.uni-kassel.de/upos.owl#Location> ;
    owl:disjointWith :BusinessPlace , :PublicPlace .

:PublicPlace
    a      owl:Class ;
    rdfs:subClassOf <http://ws.comtec.e-technik.uni-kassel.de/upos.owl#Location> ;
    owl:disjointWith :BusinessPlace , :PrivatePlace .

:BusinessPlace
    a      owl:Class ;
    rdfs:subClassOf <http://ws.comtec.e-technik.uni-kassel.de/upos.owl#Location> ;
    owl:disjointWith :PublicPlace , :PrivatePlace .

:GeopoliticalEntity
    a       owl:Class ;
    rdfs:subClassOf <http://ws.comtec.e-technik.uni-kassel.de/upos.owl#Location> .

:AccessRange
    a       owl:Class ;
    rdfs:subClassOf <http://ws.comtec.e-technik.uni-kassel.de/upos.owl#Location> .

:BtAccessRange_Business
    a       owl:Class ;
    rdfs:subClassOf :AccessRange_Business .

:Plane
    a       owl:Class ;
    rdfs:subClassOf :Vehicle_Public .

:PresentationSpace
    a       owl:Class ;
    rdfs:subClassOf :Room_Business ;
    owl:equivalentClass :VideoProjectorSpace .

:Airport
    a       owl:Class ;
    rdfs:subClassOf :TravelPoint .

:Store
    a       owl:Class ;
    rdfs:subClassOf :ShoppingPlace .

:ClassRoom
    a       owl:Class ;
    rdfs:subClassOf :Room_Educational .

:FootballGround
    a       owl:Class ;
    rdfs:subClassOf :SportsField .

:Accommodation
    a       owl:Class ;
    rdfs:subClassOf :PublicPlace .

:Church
    a       owl:Class ;
    rdfs:subClassOf :ReligiousPlace .

:Bathroom_TravelPoint
    a       owl:Class ;
    rdfs:subClassOf :Room_TravelPoint .
```

```
:Auditorium_Educational
      a       owl:Class ;
      rdfs:subClassOf :Room_Educational .

:CarPark_Business
      a       owl:Class ;
      rdfs:subClassOf :BusinessPlace .

:Home
      a       owl:Class ;
      rdfs:subClassOf :PrivatePlace .

:Workroom
      a       owl:Class ;
      rdfs:subClassOf :Room_Private .

:Storehouse
      a       owl:Class ;
      rdfs:subClassOf :Building_Business .

:Nightclub
      a       owl:Class ;
      rdfs:subClassOf :EntertainmentPlace .

:Floor_Business
      a       owl:Class ;
      rdfs:subClassOf :BusinessPlace .

:Laboratory
      a       owl:Class ;
      rdfs:subClassOf :Workspace_Business .

:SportsField
      a       owl:Class ;
      rdfs:subClassOf :RecreationArea .

:SinglePersonOffice
      a       owl:Class ;
      rdfs:subClassOf :Office .

:Camping
      a       owl:Class ;
      rdfs:subClassOf :Accommodation .

:BtAccessRange_Public
      a       owl:Class ;
      rdfs:subClassOf :AccessRange_Public .

:ThreePersonOffice
      a       owl:Class ;
      rdfs:subClassOf :Office .
```

```
:Zoo
    a      owl:Class ;
    rdfs:subClassOf :TouristPlace .

:Office
    a      owl:Class ;
    rdfs:subClassOf :Workspace_Business .

:Bus
    a      owl:Class ;
    rdfs:subClassOf :Vehicle_Public .

:Campus
    a      owl:Class ;
    rdfs:subClassOf :EducationalPlace .

:Bedroom_Private
    a      owl:Class ;
    rdfs:subClassOf :Room_Private .

:PrivateActivity
    a      owl:Class ;
    rdfs:subClassOf <http://ws.comtec.e-technik.uni-kassel.de/upos.owl#Activity> ;
    owl:disjointWith :BusinessActivity .

:Chair
    a      owl:Class ;
    rdfs:subClassOf :EducationalPlace .

:UniversityBuilding
    a      owl:Class ;
    rdfs:subClassOf :Building_Educational .

:WlanAccessRange_Business
    a      owl:Class ;
    rdfs:subClassOf :AccessRange_Business .

:Park
    a      owl:Class ;
    rdfs:subClassOf :RecreationArea .

:Port
    a      owl:Class ;
    rdfs:subClassOf :TravelPoint .

:Garden_Private
    a      owl:Class ;
    rdfs:subClassOf :PrivatePlace .

:Cafe
```

```
    a       owl:Class ;
    rdfs:subClassOf :GastronomyPlace .

:ConcertHall
    a       owl:Class ;
    rdfs:subClassOf :EntertainmentPlace .

:Room_Accommodation
    a       owl:Class ;
    rdfs:subClassOf :Accommodation .

:Theater
    a       owl:Class ;
    rdfs:subClassOf :EntertainmentPlace .

:ConventionCenter
    a       owl:Class ;
    rdfs:subClassOf :Building_Business .

:AccessRange_Public
    a       owl:Class ;
    rdfs:subClassOf :PublicPlace .

:StoreRoom
    a       owl:Class ;
    rdfs:subClassOf :Room_Private .

:Garden_Business
    a       owl:Class ;
    rdfs:subClassOf :BusinessPlace .

:Auditorium_Business
    a       owl:Class ;
    rdfs:subClassOf :Room_Business .

:Car_Private
    a       owl:Class ;
    rdfs:subClassOf :Vehicle_Private .

:Ship
    a       owl:Class ;
    rdfs:subClassOf :Vehicle_Public .

:WlanAccessRange_Private
    a       owl:Class ;
    rdfs:subClassOf :AccessRange_Private .

:VideoProjectorSpace
    a       owl:Class ;
    rdfs:subClassOf :Room_Business ;
    owl:equivalentClass :PresentationSpace .
```

```
:Building_Business
      a       owl:Class ;
      rdfs:subClassOf :BusinessPlace .

:ReadingRoom_Private
      a       owl:Class ;
      rdfs:subClassOf :Room_Private .

:Playroom_Educational
      a       owl:Class ;
      rdfs:subClassOf :Room_Educational .

:HeavyGoodsVehicle
      a       owl:Class ;
      rdfs:subClassOf :Vehicle_Business .

:Train
      a       owl:Class ;
      rdfs:subClassOf :Vehicle_Public .

:Department
      a       owl:Class ;
      rdfs:subClassOf :BusinessPlace .

:Room_Business
      a       owl:Class ;
      rdfs:subClassOf :BusinessPlace .

:Basement_Private
      a       owl:Class ;
      rdfs:subClassOf :Room_Private .

:HotelRoom
      a       owl:Class ;
      rdfs:subClassOf :Room_Accommodation .

:FactoryBuilding
      a       owl:Class ;
      rdfs:subClassOf :Building_Business .

:OfficeBuilding
      a       owl:Class ;
      rdfs:subClassOf :Building_Business .

:Guesthouse
      a       owl:Class ;
      rdfs:subClassOf :Accommodation .

:ChildrenBedroom
      a       owl:Class ;
```

```
    rdfs:subClassOf :Bedroom_Private .

:Car_Business
    a       owl:Class ;
    rdfs:subClassOf :Vehicle_Business .

:PublicAuthority
    a       owl:Class ;
    rdfs:subClassOf :AdministerialPlace .

:Gym
    a       owl:Class ;
    rdfs:subClassOf :RecreationArea .

:SchoolBuilding
    a       owl:Class ;
    rdfs:subClassOf :Building_Educational .

:Room_TravelPoint
    a       owl:Class ;
    rdfs:subClassOf :TravelPoint .

:Lounge_Business
    a       owl:Class ;
    rdfs:subClassOf :Room_Business .

:Grocery
    a       owl:Class ;
    rdfs:subClassOf :Store .

:Region
    a       owl:Class ;
    rdfs:subClassOf :GeopoliticalEntity .

:UmtsAccessRange
    a       owl:Class ;
    rdfs:subClassOf :UmtsBasedAccessRange .

:EducationalPlace
    a       owl:Class ;
    rdfs:subClassOf :PublicPlace .

:Wing_Private
    a       owl:Class ;
    rdfs:subClassOf :PrivatePlace .

:PrinterSpace_Private
    a       owl:Class ;
    rdfs:subClassOf :Room_Private .

:ConferenceRoom
```

```
      a       owl:Class ;
      rdfs:subClassOf :Workspace_Business ;
      owl:equivalentClass :MeetingRoom .

:SeminarRoom
      a       owl:Class ;
      rdfs:subClassOf :Workspace_Business ;
      owl:equivalentClass :MeetingRoom .

:Foyer_TravelPoint
      a       owl:Class ;
      rdfs:subClassOf :Room_TravelPoint .

:Bathroom_Business
      a       owl:Class ;
      rdfs:subClassOf :Room_Business .

:HsdpaAccessRange
      a       owl:Class ;
      rdfs:subClassOf :UmtsBasedAccessRange .

:RecreationArea
      a       owl:Class ;
      rdfs:subClassOf :PublicPlace .

:Lounge_Educational
      a       owl:Class ;
      rdfs:subClassOf :Room_Educational .

:Monument
      a       owl:Class ;
      rdfs:subClassOf :TouristPlace .

:Guestroom
      a       owl:Class ;
      rdfs:subClassOf :Room_Private ;
      owl:equivalentClass :GuestBedroom .

:GsmAccessRange
      a       owl:Class ;
      rdfs:subClassOf :GsmBasedAccessRange .

:Wing_Business
      a       owl:Class ;
      rdfs:subClassOf :BusinessPlace .

:TravelPoint
      a       owl:Class ;
      rdfs:subClassOf :PublicPlace .

:AccessRange_Private
```

```
    a       owl:Class ;
    rdfs:subClassOf :PrivatePlace .

:Bedroom_Accommodation
    a       owl:Class ;
    rdfs:subClassOf :Room_Accommodation .

:Floor_Private
    a       owl:Class ;
    rdfs:subClassOf :PrivatePlace .

:GsmBasedAccessRange
    a       owl:Class ;
    rdfs:subClassOf :AccessRange .

:GprsAccessRange
    a       owl:Class ;
    rdfs:subClassOf :GsmBasedAccessRange .

:SummerHouse
    a       owl:Class ;
    rdfs:subClassOf :PrivatePlace .

:Continent
    a       owl:Class ;
    rdfs:subClassOf :GeopoliticalEntity .

:CityHall
    a       owl:Class ;
    rdfs:subClassOf :AdministerialPlace .

:Room_Educational
    a       owl:Class ;
    rdfs:subClassOf :EducationalPlace .

:Country
    a       owl:Class ;
    rdfs:subClassOf :GeopoliticalEntity .

:PostOffice
    a       owl:Class ;
    rdfs:subClassOf :AdministerialPlace .

:Eating_Private
    a       owl:Class ;
    rdfs:subClassOf :PrivateActivity .

:ShoppingMall
    a       owl:Class ;
    rdfs:subClassOf :ShoppingPlace .
```

```
:Hotel
    a       owl:Class ;
    rdfs:subClassOf :Accommodation .

:Working_Private
    a       owl:Class ;
    rdfs:subClassOf :PrivateActivity .

:PoliceStation
    a       owl:Class ;
    rdfs:subClassOf :AdministerialPlace .

:Motorbike_Private
    a       owl:Class ;
    rdfs:subClassOf :Vehicle_Private .

:Garage_Private
    a       owl:Class ;
    rdfs:subClassOf :PrivatePlace .

:Mosque
    a       owl:Class ;
    rdfs:subClassOf :ReligiousPlace .

:MeetingRoom
    a       owl:Class ;
    rdfs:subClassOf :Workspace_Business ;
    owl:equivalentClass :ConferenceRoom , :SeminarRoom .

:Vehicle_Business
    a       owl:Class ;
    rdfs:subClassOf :BusinessPlace .

:Stairway_Private
    a       owl:Class ;
    rdfs:subClassOf :PrivatePlace .

:Vehicle_Private
    a       owl:Class ;
    rdfs:subClassOf :PrivatePlace .

:Vehicle_Public
    a       owl:Class ;
    rdfs:subClassOf :PublicPlace .

:PublicLibrary
    a       owl:Class ;
    rdfs:subClassOf :AdministerialPlace .

:Disco
    a       owl:Class ;
```

```
      rdfs:subClassOf :EntertainmentPlace .

:GolfCourse
      a       owl:Class ;
      rdfs:subClassOf :SportsField .

:Aquarium
      a       owl:Class ;
      rdfs:subClassOf :TouristPlace .

:Bathroom_Private
      a       owl:Class ;
      rdfs:subClassOf :Room_Private .

:PrinterSpace_Business
      a       owl:Class ;
      rdfs:subClassOf :Room_Business .

:Restaurant
      a       owl:Class ;
      rdfs:subClassOf :GastronomyPlace .

:AdministerialPlace
      a       owl:Class ;
      rdfs:subClassOf :PublicPlace .

:Tram
      a       owl:Class ;
      rdfs:subClassOf :Vehicle_Public .

:DoingSports
      a       owl:Class ;
      rdfs:subClassOf :PrivateActivity .

:Market
      a       owl:Class ;
      rdfs:subClassOf :ShoppingPlace .

:Room_Private
      a       owl:Class ;
      rdfs:subClassOf :PrivatePlace .

:Premises
      a       owl:Class ;
      rdfs:subClassOf :BusinessPlace .

:Exhibition
      a       owl:Class ;
      rdfs:subClassOf :TouristPlace .

:EquipmentRoom_Business
```

```
      a       owl:Class ;
      rdfs:subClassOf :Room_Business .

:Bicycle_Private
      a       owl:Class ;
      rdfs:subClassOf :Vehicle_Private .

:Temple
      a       owl:Class ;
      rdfs:subClassOf :ReligiousPlace .

:BusinessActivity
      a       owl:Class ;
      rdfs:subClassOf <http://ws.comtec.e-technik.uni-kassel.de/upos.owl#Activity> ;
      owl:disjointWith :PrivateActivity .

:Workspace_Business
      a       owl:Class ;
      rdfs:subClassOf :Room_Business .

:Library_Private
      a       owl:Class ;
      rdfs:subClassOf :Room_Private .

:OpenPlanOffice
      a       owl:Class ;
      rdfs:subClassOf :Office .

:Hospital
      a       owl:Class ;
      rdfs:subClassOf :HealthCarePlace .

:Building_Educational
      a       owl:Class ;
      rdfs:subClassOf :EducationalPlace .

:Motorbike_Business
      a       owl:Class ;
      rdfs:subClassOf :Vehicle_Business .

:Cab
      a       owl:Class ;
      rdfs:subClassOf :Vehicle_Public .

:SummerCottage
      a       owl:Class ;
      rdfs:subClassOf :PrivatePlace .

:Bathroom_Educational
      a       owl:Class ;
      rdfs:subClassOf :Room_Educational .
```

```
:DiningRoom_Private
    a       owl:Class ;
    rdfs:subClassOf :Room_Private .

:SchoolCampus
    a       owl:Class ;
    rdfs:subClassOf :Campus .

:ShoppingPlace
    a       owl:Class ;
    rdfs:subClassOf :PublicPlace .

:GastronomyPlace
    a       owl:Class ;
    rdfs:subClassOf :PublicPlace .

:Kitchen_Business
    a       owl:Class ;
    rdfs:subClassOf :Room_Business .

:Lounge_TravelPoint
    a       owl:Class ;
    rdfs:subClassOf :Room_TravelPoint .

:EdgeAccessRange
    a       owl:Class ;
    rdfs:subClassOf :GsmBasedAccessRange .

:Foyer_Business
    a       owl:Class ;
    rdfs:subClassOf :Room_Business .

:Metro
    a       owl:Class ;
    rdfs:subClassOf :Vehicle_Public .

:ReligiousPlace
    a       owl:Class ;
    rdfs:subClassOf :PublicPlace .

:Hostel
    a       owl:Class ;
    rdfs:subClassOf :Accommodation .

:Library_Business
    a       owl:Class ;
    rdfs:subClassOf :Room_Business .

:Kindergarten
    a       owl:Class ;
```

```
      rdfs:subClassOf :EducationalPlace .

:WlanAccessRange_Public
      a       owl:Class ;
      rdfs:subClassOf :AccessRange_Public .

:Stairway_Business
      a       owl:Class ;
      rdfs:subClassOf :BusinessPlace .

:MasterBedroom
      a       owl:Class ;
      rdfs:subClassOf :Bedroom_Private .

:Reception_Business
      a       owl:Class ;
      rdfs:subClassOf :Room_Business .

:TennisCourt
      a       owl:Class ;
      rdfs:subClassOf :SportsField .

:UniversityCampus
      a       owl:Class ;
      rdfs:subClassOf :Campus .

:GuestBedroom
      a       owl:Class ;
      rdfs:subClassOf :Bedroom_Private ;
      owl:equivalentClass :Guestroom .

:BtAccessRange_Private
      a       owl:Class ;
      rdfs:subClassOf :AccessRange_Private .

:DiningRoom_Accommodation
      a       owl:Class ;
      rdfs:subClassOf :Room_Accommodation .

:Opera
      a       owl:Class ;
      rdfs:subClassOf :EntertainmentPlace .

:HighschoolBuilding
      a       owl:Class ;
      rdfs:subClassOf :Building_Educational .

:AccessRange_Business
      a       owl:Class ;
      rdfs:subClassOf :BusinessPlace .
```

```
:SportsCentre
      a       owl:Class ;
      rdfs:subClassOf :RecreationArea .

:Pastime
      a       owl:Class ;
      rdfs:subClassOf :PrivateActivity .

:EntertainmentPlace
      a       owl:Class ;
      rdfs:subClassOf :PublicPlace .

:LectureRoom
      a       owl:Class ;
      rdfs:subClassOf :Room_Educational .

:LivingRoom
      a       owl:Class ;
      rdfs:subClassOf :Room_Private .

:Corridor_Private
      a       owl:Class ;
      rdfs:subClassOf :Room_Private .

:Corridor_Business
      a       owl:Class ;
      rdfs:subClassOf :Room_Business .

:Cinema
      a       owl:Class ;
      rdfs:subClassOf :EntertainmentPlace .

:SwimmingPool
      a       owl:Class ;
      rdfs:subClassOf :RecreationArea .

:TwoPersonOffice
      a       owl:Class ;
      rdfs:subClassOf :Office .

:TouristPlace
      a       owl:Class ;
      rdfs:subClassOf :PublicPlace .

:Museum
      a       owl:Class ;
      rdfs:subClassOf :TouristPlace .

:UmtsBasedAccessRange
      a       owl:Class ;
      rdfs:subClassOf :AccessRange .
```

```
:Bistro
    a       owl:Class ;
    rdfs:subClassOf :GastronomyPlace .

:MedicalPractice
    a       owl:Class ;
    rdfs:subClassOf :HealthCarePlace .

:HealthCarePlace
    a       owl:Class ;
    rdfs:subClassOf :PublicPlace .

:Bar
    a       owl:Class ;
    rdfs:subClassOf :GastronomyPlace .

:City
    a       owl:Class ;
    rdfs:subClassOf :GeopoliticalEntity .

:HighschoolCampus
    a       owl:Class ;
    rdfs:subClassOf :Campus .

:Station
    a       owl:Class ;
    rdfs:subClassOf :TravelPoint .

:DepartmentStore
    a       owl:Class ;
    rdfs:subClassOf :Store .

:Bathroom_Accommodation
    a       owl:Class ;
    rdfs:subClassOf :Room_Accommodation .

:Faculty
    a       owl:Class ;
    rdfs:subClassOf :EducationalPlace .

:ThemePark
    a       owl:Class ;
    rdfs:subClassOf :TouristPlace .

:SportsArena
    a       owl:Class ;
    rdfs:subClassOf :EntertainmentPlace .

:Secretariat_Business
    a       owl:Class ;
    rdfs:subClassOf :Room_Business .

:Kitchen_Private
    a       owl:Class ;
    rdfs:subClassOf :Room_Private .

:Playroom
    a       owl:Class ;
    rdfs:subClassOf :Room_Private .
```

Appendix F: Measurement Results

Table 10 shows the results for the measurement step 1, i.e. the creation of the combined ontology model. In this step, first an empty Jena2 [24] [25] DefaultModel is created. Afterwards, the user profile ontology introduced in section 4.2, the context ontology introduced in section 4.3 and the user-specific Individuals Database introduced in section 5.1 are separately read from the file system, and added to the combined ontology model. This process, as well as the computations for the arithmetic mean $\bar{x}$, the range *R* and the standard deviation *s*, is described in more detail in section 6.1.3.

Table 10: Results for the Measurement Step 1 in Milliseconds

Measurement Step ------- ***Approach***	***Ontology Loading Steps (Arithmetic Means $\bar{x}$ in ms)***			***Sum of Ontology Loading Steps***		
	Creating Jena2 Default Model and Loading User Profile Ontology	***Loading and Adding Location Ontology***	***Loading and Adding Individuals Database***	***Arithmetic Mean $\bar{x}$ in ms***	***Range R in ms***	***Standard Deviation s in ms***
Classification – Standard Version	873	140	92	1105	113	32.1
Ontology – Standard Version	990	135	197	1322	77	23.4
Classification – 10x Version	885	139	397	1421	108	32.3
Ontology – 10x-A Version	973	136	817	1926	94	27.6
Ontology – 10x-B Version	983	133	833	1949	296	81.5
Classification – 100x Version	860	147	2192	3199	156	45.9
Ontology – 100x-A Version	977	128	6614	7719	78	20.8
Ontology – 100x-B Version	983	130	6277	7390	93	33.0

Table 10 includes the results for the following variants of the user profile selection mechanism:

Classification – Standard Version: User profile ontology as introduced in section 4.2 without any properties, i.e. only classification, location ontology as introduced in section 4.3 without any properties, i.e. only classification, and Individuals Database as introduced in section 6.1.2 without any relationships between location instances as described in section 6.2.5.

Ontology – Standard Version: User profile ontology as introduced in section 4.2, location ontology as introduced in section 4.3 and Individuals Database as introduced in section 6.1.2.

Classification – 10x Version: Unlike the Classification – Standard Version above, this variation includes an Individuals Database with 10 times as many individuals as described in section 6.2.5.

Ontology – 10x-A Version: Unlike the Ontology – Standard Version above, this variation includes an Individuals Database with 10 times as many individuals. The interrelations between these individuals is described by variant A in section 6.2.3 and 6.2.4.

Ontology – 10x-B Version: Unlike the Ontology – Standard Version above, this variation includes an Individuals Database with 10 times as many individuals. The interrelations between these individuals is described by variant B in section 6.2.3 and 6.2.4.

Classification – 100x Version: Unlike the Classification – Standard Version above, this variation includes an Individuals Database with 100 times as many individuals as described in section 6.2.5.

Ontology – 100x-A Version: Unlike the Ontology – Standard Version above, this variation includes an Individuals Database with 100 times as many individuals. The interrelations between these individuals is described by variant A in section 6.2.3 and 6.2.4.

Ontology – 100x-B Version: Unlike the Ontology – Standard Version above, this variation includes an Individuals Database with 100 times as many individuals. The interrelations between these individuals is described by variant B in section 6.2.3 and 6.2.4.

Table 11: Results for Figure 43 and Figure 44 in Milliseconds

Reasoner ------- ***Query Sample***			***Pellet Config 1***	***Pellet Config 2***	***Jena2***	***KAON2***	***Pellet DIG***	***FaCT++ DIG***
A1	***Query Only***	$\bar{x}$	110	456	886	1591	1180	1075
		R	31	31	16	157	172	32
		s	7.3	9.6	7.5	44.3	52.4	14.4
	All	$\bar{x}$	897	1142	1335	2941	1633	1523
		R	63	77	32	125	157	47
		s	18.3	25.9	11.0	35.3	53.3	18.5
B1	***Query Only***	$\bar{x}$	152	471	1472	1653	1273	n/a
		R	32	32	32	156	219	n/a
		s	10.8	11.5	10.0	54.9	77.3	n/a
	All	$\bar{x}$	943	1157	1921	3014	1727	n/a
		R	62	95	31	170	219	n/a
		s	22.0	29.7	11.4	58.2	80.0	n/a

C1	***Query Only***	$\bar{x}$	131	473	1306	1642	1246	n/a
		R	32	17	32	156	188	n/a
		s	11.1	7.7	10.9	57.6	67.1	n/a
	All	$\bar{x}$	919	1159	1755	2994	1700	n/a
		R	63	61	32	140	204	n/a
		s	16.3	21.7	12.6	57.4	68.0	n/a
D1	***Query Only***	$\bar{x}$	119	469	917	1639	1222	1100
		R	16	47	32	125	202	47
		s	8.0	16.3	10.8	50.4	74.8	16.7
	All	$\bar{x}$	908	1155	1366	2986	1675	1548
		R	32	79	47	124	202	78
		s	15.3	27.1	15.2	40.6	76.5	23.6
E1	***Query Only***	$\bar{x}$	187	784	1287	1603	1399	n/a
		R	31	156	48	47	344	n/a
		s	12.7	53.0	15.5	19.8	108.6	n/a
	All	$\bar{x}$	977	1470	1736	2951	1852	n/a
		R	62	203	49	61	265	n/a
		s	25.3	65.1	14.1	24.9	104.6	n/a
F1	***Query Only***	$\bar{x}$	180	536	1261	1552	1301	n/a
		R	32	47	47	47	313	n/a
		s	13.3	14.8	14.9	16.6	107.6	n/a
	All	$\bar{x}$	975	1222	1710	2900	1755	n/a
		R	62	95	47	77	314	n/a
		s	19.7	35.2	13.2	24.7	108.5	n/a

Table 11 shows the detailed measurement results in milliseconds for the summary shown in Figure 43 and Figure 44. The computations for the arithmetic mean $\bar{x}$, the range *R* and the standard deviation *s* are described in more detail in section 6.1.3.

Table 12: Results for Figure 48 and Figure 49 in Milliseconds

Reasoner ------- Query Sample			***Pellet Config 1***	***Pellet Config 2***	***Jena2***	***KAON2***	***Pellet DIG***	***FaCT++ DIG***
A1	***Query Only***	$\bar{x}$	92	889	1223	2702	3502	3125
		R	31	78	187	313	562	156
		s	11.5	22.6	72.5	124.2	200.9	51.0
	All	$\bar{x}$	2183	2481	2572	6398	4775	4402
		R	62	94	187	359	593	172
		s	25.3	28.5	67.5	126.2	205.8	50.5
B1	***Query Only***	$\bar{x}$	206	896	1727	2731	3683	n/a
		R	16	16	63	250	478	n/a
		s	9.8	14.9	22.6	105.1	152.6	n/a
	All	$\bar{x}$	2300	2488	3075	6427	4957	n/a
		R	125	48	79	328	494	n/a
		s	42.2	16.4	32.3	114.6	155.4	n/a
C1	***Query Only***	$\bar{x}$	198	912	1611	2749	3668	n/a
		R	31	63	32	250	649	n/a
		s	12.7	23.4	13.9	82.7	242.8	n/a
	All	$\bar{x}$	2294	2505	2960	6442	4942	n/a

		R	47	48	94	157	648	n/a
		s	17.6	22.3	26.8	78.6	245.1	n/a
D1	***Query Only***	$\bar{x}$	155	903	1347	2703	5344	3198
		R	31	78	94	344	563	156
		s	11.6	23.1	29.4	104.5	347.1	53.6
	All	$\bar{x}$	2250	2495	2696	6409	4817	4475
		R	78	63	126	219	562	156
		s	26.6	21.1	37.8	105.9	346.9	58.2
E1	***Query Only***	$\bar{x}$	261	2226	1591	2852	3666	n/a
		R	31	140	147	313	468	n/a
		s	10.5	41.1	41.6	109.3	197.5	n/a
	All	$\bar{x}$	2358	3819	2940	6552	4939	n/a
		R	80	156	209	329	484	n/a
		s	24.1	43.7	58.2	111.7	200.3	n/a
F1	***Query Only***	$\bar{x}$	250	1210	1591	2805	3719	n/a
		R	31	220	94	329	500	n/a
		s	10.5	82.6	33.5	120.7	155.5	n/a
	All	$\bar{x}$	2335	2802	2940	6509	4992	n/a
		R	94	204	93	345	515	n/a
		s	28.6	77.9	36.4	123.1	159.5	n/a

Table 12 shows the detailed measurement results in milliseconds for the summary shown in Figure 48 and Figure 49. The computations for the arithmetic mean $\bar{x}$, the range *R* and the standard deviation *s* are described in more detail in section 6.1.3.

Table 13: Results for Figure 50 and Figure 51 in Milliseconds

Reasoner ------- Query Sample			***Pellet Config 1***	***Pellet Config 2***	***Jena2***	***KAON2***	***FaCT++ DIG***
A1	***Query Only***	$\bar{x}$	1022	2830	4111	15689	23585
		R	47	94	391	1328	4174
		s	16.9	28.2	135.9	450.9	1920.0
	All	$\bar{x}$	14130	13203	14335	40030	34419
		R	359	218	911	1359	6375
		s	113.2	58.2	272.2	464.2	1913.4
D1	***Query Only***	$\bar{x}$	1100	2874	4109	13884	27090
		R	47	64	421	1187	26402
		s	15.2	22.8	147.3	363.3	8245.8
	All	$\bar{x}$	14234	13247	14354	38233	37924
		R	249	204	882	905	26356
		s	81.9	58.1	290.9	371.3	8225.4

Table 13 shows the detailed measurement results in milliseconds for the summary shown in Figure 50 and Figure 51. The computations for the arithmetic mean $\bar{x}$, the range *R* and the standard deviation *s* are described in more detail in section 6.1.3.

Table 14: Results for Figure 53 and Figure 54 in Milliseconds

Reasoner ------- *Query Sample*			*Pellet Config 1*	*Pellet Config 2*	*Jena2*	*KAON2*	*Pellet DIG*	*FaCT++ DIG*
A1	***Query Only***	$\bar{x}$	100	894	1150	24008	3469	3135
		R	32	47	17	4781	672	188
		s	11.0	14.4	8.2	1616.2	243.3	62.6
	All	$\bar{x}$	2181	2484	2406	27694	4727	4403
		R	64	110	33	3829	532	173
		s	23.7	40.4	12.8	1617.1	244.8	63.0
B1	***Query Only***	$\bar{x}$	261	901	3717	23525	3659	n/a
		R	48	63	94	3377	546	n/a
		s	18.3	18.3	35.1	1553.6	199.6	n/a
	All	$\bar{x}$	2345	2491	4973	27210	4917	n/a
		R	94	163	94	3298	578	n/a
		s	24.9	46.1	33.8	1558.5	205.1	n/a
C1	***Query Only***	$\bar{x}$	331	909	Error	23580	3613	n/a
		R	63	63	Error	3999	626	n/a
		s	21.8	19.2	Error	1577.5	220.2	n/a
	All	$\bar{x}$	2407	2499	Error	27253	4871	n/a
		R	94	116	Error	3936	657	n/a
		s	29.1	35.8	Error	1562.3	228.2	n/a
D1	***Query Only***	$\bar{x}$	151	906	1192	24230	3592	3156
		R	31	63	32	4906	594	62
		s	10.5	24.6	12.9	1923.2	222.2	23.2
	All	$\bar{x}$	2239	2496	2449	27912	4850	4425
		R	62	131	63	5001	579	109
		s	24.4	48.5	19.6	1918.1	226.0	32.6
E1	***Query Only***	$\bar{x}$	323	2202	1581	24488	3686	n/a
		R	47	125	78	5459	453	n/a
		s	19.5	37.2	27.2	1527.8	199.6	n/a
	All	$\bar{x}$	2407	3791	2838	28178	4944	n/a
		R	78	130	78	5538	484	n/a
		s	22.6	52.1	25.7	1553.4	206.1	n/a
F1	***Query Only***	$\bar{x}$	392	1697	Error	25445	3672	n/a
		R	78	188	Error	3469	642	n/a
		s	24.0	60.2	Error	1163.2	224.1	n/a
	All	$\bar{x}$	2468	3287	Error	29124	4930	n/a
		R	119	281	Error	3501	642	n/a
		s	39.5	87.6	Error	1171.8	226.6	n/a

Table 14 shows the detailed measurement results in milliseconds for the summary shown in Figure 53 and Figure 54. The computations for the arithmetic mean $\bar{x}$, the range R and the standard deviation s are described in more detail in section 6.1.3.

Table 15: Results for Figure 55 and Figure 56 in Milliseconds

Reasoner ------- *Query Sample*			*Pellet Config 1*	*Pellet Config 2*	*Jena2*	*FaCT++ DIG*
A1	***Query Only***	$\bar{x}$	1163	4141	4407	25783
		R	78	141	615	23000
		s	23.55	42.3	176.1	7393.4
	All	$\bar{x}$	14706	14230	14274	35782
		R	204	189	614	22922
		s	65.6	68.0	178.6	7441.6
D1	***Query Only***	$\bar{x}$	1253	4148	4502	21817
		R	16	78	500	12594
		s	6.5	22.4	182.9	3863.5
	All	$\bar{x}$	14816	14237	14369	31817
		R	140	265	484	12547
		s	61.6	75.0	180.0	3846.8

Table 15 shows the detailed measurement results in milliseconds for the summary shown in Figure 55 and Figure 56. The computations for the arithmetic mean $\bar{x}$, the range R and the standard deviation s are described in more detail in section 6.1.3.

Table 16: Results for Figure 59 and Figure 60 (Classification Standard Version)

Reasoner ------- *Query Sample*			*Pellet Config 1*	*Pellet Config 2*	*Jena2*	*KAON2*	*Pellet DIG*
A1	***Query Only***	$\bar{x}$	94	386	773	655	950
		R	31	63	16	32	219
		s	7.3	18.2	8.1	9.0	64.2
	All	$\bar{x}$	769	1109	1202	1644	1374
		R	77	79	47	78	250
		s	25.0	31.3	18.6	24.2	73.0
D1	***Query Only***	$\bar{x}$	119	388	802	667	976
		R	16	62	47	32	186
		s	8.3	17.6	12.9	10.6	63.5
	All	$\bar{x}$	789	1111	1230	1659	1400
		R	78	93	63	62	186
		s	24.6	34.7	21.1	19.0	72.8

Table 16, Table 17 and Table 18 shows the detailed measurement results in milliseconds for the summary shown in Figure 59 and Figure 60. The computations for the arithmetic mean $\bar{x}$, the range R and the standard deviation s are described in more detail in section 6.1.3.

Table 17: Results for Figure 59 and Figure 60 (Classification 10x Version)

Reasoner ------- ***Query Sample***			***Pellet Config 1***	***Pellet Config 2***	***Jena2***	***KAON2***	***Pellet DIG***
A1	***Query Only***	$\bar{x}$	102	811	839	689	2205
		R	17	63	31	17	531
		s	8.3	20.2	14.7	8.4	178.9
	All	$\bar{x}$	1347	1738	1520	2784	2898
		R	48	62	62	32	532
		s	14.6	21.8	24.3	12.3	179.7
D1	***Query Only***	$\bar{x}$	153	805	877	648	2242
		R	32	17	32	17	531
		s	10.1	8.6	8.9	8.3	191.4
	All	$\bar{x}$	1409	1731	1558	2793	2936
		R	78	48	31	62	516
		s	25.6	14.6	10.4	19.1	191.6

Table 18: Results for Figure 59 and Figure 60 (Classification 100x Version)

Reasoner ------- ***Query Sample***			***Pellet Config 1***	***Pellet Config 2***	***Jena2***	***KAON2***	***Pellet DIG***
A1	***Query Only***	$\bar{x}$	664	2003	1997	1300	10505
		R	32	62	48	78	1656
		s	11.1	27.2	19.4	28.5	710.7
	All	$\bar{x}$	6381	5937	5909	11991	14217
		R	130	141	95	125	1626
		s	37.6	50.4	36.2	40.4	710.9
D1	***Query Only***	$\bar{x}$	741	2014	2010	1311	10708
		R	32	31	31	47	1500
		s	11.0	8.8	11.0	15.6	459.7
	All	$\bar{x}$	6534	5948	5922	11901	14420
		R	126	111	125	1016	1516
		s	50.7	37.8	36.9	312.1	460.0

List of Figures

List of Tables

Acronyms

3G	3rd Generation
3GPP	3rd Generation Partnership Project
CC/PP	Composite Capabilities/Preference Profiles
DAML	DARPA Agent Markup Language
DARPA	Defense Advanced Research Projects Agency
DIG	Description Logic Implementation Group
EDGE	Enhanced Data Rates for GSM Evolution
ETSI	European Telecommunications Standards Institute
FOAF	Friend of a Friend
GERAN	GSM EDGE Radio Access Network
GPS	Global Positioning System
GSM	Global System for Mobile Communications
GUP	Generic User Profile
HTML	Hypertext Markup Language
HTTP	Hypertext Transfer Protocol
IP	Internet Protocol
N3	Notation 3
NS	Namespace
OMA	Open Mobile Alliance
OWL	Web Ontology Language
OWL DL	OWL Description Logics
PSE	Personal Service Environment
QoS	Quality of Service
RDF	Resource Description Framework
RDFS	RDF Schema
SPICE	Service Platform for Innovative Communication Environment

SWRL	Semantic Web Rule Language
UAProf	User Agent Profile
UMTS	Universal Mobile Telecommunications System
UPOS	User Profile Ontology with Situation-Dependent Preferences Support
URI	Uniform Resource Identifier
UTRAN	UMTS Terrestrial Radio Access Network
VHE	Virtual Home Environment
W3C	World Wide Web Consortium
WLAN	Wireless Local Area Network
XML	Extensible Markup Language
XMLNS	XML Namespace
XSD	XML Schema Definition

References

[1] G. Antoniou, F. van Harmelen, “A Semantic Web Primer”, ISBN 0-262-01242-1, MIT Press, 2008.

[2] J. Davies, “Semantic Web Technologies – Trends and Research in Ontology-Based Systems”, ISBN 0470025964, Wiley & Sons, 2006.

[3] A.K. Dey, “Providing Architectural Support for Building Context-Aware Applications”, PhD thesis, Georgia Institute of Technology, 2000.

[4] B. Mrohs, S. Steglich, M. Klemettinen, J.T. Salo, A. Aftelak, C. Cordier, F. Carrez, “MobiLife Service Infrastructure and SPICE Architecture Principles”, IEEE 64th VTC-2006 Fall, September 2006.

[5] C. Cordier, F. Carrez, H. van Kranenburg, C. Licciardi, J. Van Der Meer, A. Spedalieri, J.-P. Le Rouzic, J. Zoric, “Addressing the Challenges of B3G Service Delivery: the SPICE Service Platform”, 6th International Workshop on Applications and Services in Wireless Networks (ASWN 2006), ISBN: 3-8167-7111-4, Berlin, Germany, May 2006.

[6] H. Demeter, E. Kovacs, M. Shiaa, M. Boussard, A. Tarlano, R. Seidl, G. Marton, R. Kernchen, J.Rovira Simon, “Service Platform B3G – SPICE”, ICT Mobile Summit 2008, Stockholm, Sweden, June 2008.

[7] M. Klemettinen (Editor), “Enabling Technologies for Mobile Services: The MobiLife Book”, John Wiley & Sons, ISBN 0-470-51290-3, September 2007.

[8] T. Berners-Lee, J. Hendler, O. Lassila, “The Semantic Web“, The Scientific American, May 2001.

[9] T. Berners-Lee, “Semantic Web – XML 2000”, XML 2000 Conference, Washingtom DC, USA, December 2000. Available: http://www.w3.org/2000/Talks/1206-xml2k-tbl/, 9. June 2008.

[10] M. Dürst, A. Freytag, “Unicode in XML and other Markup Languages”, http://www.w3.org/TR/unicode-xml/, W3C Working Group Note, 16 May 2007.

[11] URI Planning Interest Group W3C/IETF, “URIs, URLs, and URNs: Clarifications and Recommendations 1.0”, http://www.w3.org/TR/uri-clarification/, W3C Note, 21 September 2001.

[12] A.J. Gerber, A. Barnard, A.J. van der Merwe, “Towards a Semantic Web Layered Architecture”, Proceedings of the 25th conference on IASTED International Multi-Conference: Software Engineering, Innsbruck, Austria, 2007.

[13] T. Bray, J. Paoli, C.M. Sperberg-McQueen, E. Maler, F. Yergeau, J. Cowan (Editors), “Extensible Markup Language (XML) 1.1 (Second Edition)”, http://www.w3.org/TR/xml11/, W3C Recommendation, 16 August 2006.

[14] T. Bray, D. Hollander, A. Layman, R. Tobin (Editors), “Namespaces in XML 1.1

(Second Edition)”, http://www.w3.org/TR/xml-names11/, W3C Recommendation, 16 August 2006.

[15] D.C. Fallside, P. Walmsley (Editors), “XML Schema Part 0: Primer Second Edition”, http://www.w3.org/TR/xmlschema-0/, W3C Recommendation 28 October 2004.

[16] D. Raggett, A. Le Hors, I. Jacobs (Editors), “HTML 4.01 Specification”, http://www.w3.org/TR/html401/, W3C Recommendation, 24 December 1999.

[17] G. Klyne, J.J. Carroll, B. McBride (Editors), “Resource Description Framework (RDF): Concepts and Abstract Syntax”, http://www.w3.org/TR/rdf-concepts/, W3C Recommendation, 10 February 2004.

[18] D. Brickley, R.V. Guha, B. McBride (Editors), “RDF Vocabulary Description Language 1.0: RDF Schema”, http://www.w3.org/TR/rdf-schema/, W3C Recommendation, 10 February 2004.

[19] T. Gruber, “Toward Principles for the Design of Ontologies Used for Knowledge Sharing”, International Journal Human-Computer Studies Vol. 43, Issues 5-6, November 1995.

[20] M.K. Smith, C. Welty, D.L. McGuinness (Editors), “OWL Web Ontology Language Guide”, http://www.w3.org/TR/owl-guide/, W3C Recommendation, 10 February 2004.

[21] I. Horrocks, B. Parsia, P. Patel-Schneider, J. Hendler, “Semantic Web Architecture: Stack or Two Towers?“, In Francois Fages and Sylvain Soliman, editors, Principles and Practice of Semantic Web Reasoning (PPSWR 2005), number 3703 in LNCS, pages 37-41. Springer, 2005.

[22] N. Shadbolt, W. Hall, T. Berners-Lee, “The Semantic Web Revisited”, IEEE Intelligent Systems, pp. 96-101, June 2006.

[23] E. Prud'hommeaux, A. Seaborne (Editors), “SPARQL Query Language for RDF”, http://www.w3.org/TR/rdf-sparql-query/, W3C Recommendation, 15 January 2008.

[24] B. McBride, “Jena: Implementing the RDF Model and Syntax Specification”, Proceedings of the Second International Workshop on the Semantic Web, Hongkong, China, 2001.

[25] K. Wilkinson, C. Sayers, H.A. Kuno, D. Reynolds, “Efficient RDF Storage and Retrieval in Jena2”, Proceedings of SWDB'03, The first International Workshop on Semantic Web and Databases, Berlin, Germany, September 2003.

[26] E. Bozsak et al., “KAON: Towards a large scale Semantic Web”, In proceedings of the 3rd International Conference on E-Commerce and Web Technologies (EC-Web), September 2002.

[27] B. Motik, and U. Sattler, “A Comparison of Reasoning Techniques for Querying Large Description Logic ABoxes”, Proceedings of the 13th International Conference on Logic for Programming Artificial Intelligence and Reasoning

(LPAR 2006), 2006.

[28] S. Bechhofer, R. Volz and P. Lord, "Cooking the Semantic Web with the OWL API", Proceedings of ISWC 2003, Sanibel Island, Florida, USA, October 2003.

[29] J. Broekstra, A. Kampman, F. van Harmelen, “Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema”, International Semantic Web Conference 2002, Sardinia, Italy, 2002.

[30] E. Sirin, B. Parsia, B.C. Grau, A. Kalyanpur, Y. Katz, “Pellet: A Practical OWL-DL Reasoner”, Journal of Web Semantics, Volume 5, Issue 2, June 2007.

[31] Ian Horrocks, “The FaCT System”, Proceedings of the 2nd Int. Conference on Analytic Tableaux and Related Methods, pages 307-312, Springer, 1998.

[32] D. Tsarkov, I. Horrocks, “FaCT++ Description Logic Reasoner: System Description”, In Proceedings of the Int. Joint Conference on Automated Reasoning (IJCAR 2006), 2006.

[33] V. Haarslev, R. Möller: “Description of the RACER System and its Applications”, In Proceedings of International Workshop on Description Logics (DL-2001), Stanford, USA, 1.-3. August, pages 131–141, 2001.

[34] V. Haarslev, R. Möller: “Racer: An OWL Reasoning Agent for the Semantic Web”, In Proceedings of the International Workshop on Applications, Products and Services of Web-based Support Systems, pages 91–95, Halifax, Canada, October, 2003.

[35] European Telecommunications Standards Institute (ETSI): “Human Factors (HF); User Profile Management,” ETSI EG 202 325 v1.1.1, http://www.etsi.org/, Oct 2005.

[36] 3rd Generation Partnership Project (3GPP), “Generic User Profile (GUP), Data Description Method (DDM)”, 3GPP TR 23 941 v6.0.0, http://www.3gpp.org/, Dec 2004.

[37] 3rd Generation Partnership Project (3GPP), “Generic User Profile (GUP), Architecture (Stage 2)”, 3GPP TR 23 240 v6.7.0, http://www.3gpp.org/, March 2005.

[38] G. Klyne, F. Reynolds, C. Woodrow, H. Ohto, J. Hjelm, M.H. Butler, L. Tran (Editors), “Composite Capabilities/Preference Profiles (CC/PP): Structure and Vocabularies 1.0”, http://www.w3.org/TR/CCPP-struct-vocab/, W3C Recommendation, 15 January 2004.

[39] Open Mobile Alliance (OMA), “User Agent Profile (UAProf) v2.0”, http://www.openmobilealliance.org/, May 2003.

[40] L. Suryanarayana, J. Hjelm, “Profiles for the Situated Web”, International World Wide Web Conference, Honolulu, Hawaii, USA, 2002.

[41] D. Brickley, L. Miller, “FOAF Vocabulary Specification”, http://xmlns.com/foaf/0.1/, July 2005.

[42] Internet Engineering Task Force (IETF), “vCard MIME Directory Profile,” RFC 2426, http://www.ietf.org/rfc/rfc2426.txt, September 1998.

[43] J. Orwant, “Heterogeneous Learning in the Doppelgänger User Modeling System”, User Modeling and User-Adapted Interaction Journal, 4:107–130, 1994.

[44] D. Heckmann, T. Schwartz, B. Brandherm, M. Schmitz, and M. v. Wilamowitz-Moellendorff, “GUMO – The General User Model Ontology”, Proceedings of the 10th International Conference on User Modeling (UM'2005), Edinburgh, UK, 2005.

[45] D. Heckmann, T. Schwartz, B. Brandherm, A. Kröner, “Decentralized User Modeling with UserML and GUMO”, Proceedings of the Workshop on Decentralized, Agent Based and Social Approaches to User Modelling (DASUM 2005), Edinburgh, UK, pp. 61-65, 2005.

[46] D. Heckmann, “Ubiquitous User Modeling”, Dissertations in Artificial Intelligence-Infix, Volume 297, IOS Press, The Netherlands, October 2006.

[47] M. Golemati, A. Katifori, C. Vassilakis, G. Lepouras, C. Halatsis, “Creating an Ontology for the User Profile: Method and Applications”, First IEEE International Conference on Research Challenges in Information Science (RCIS), Morocco 2007.

[48] S. Melnik, H. Garcia-Molina, E. Rahm, “Similarity Flooding: A Versatile Graph Matching Algorithm and its Application to Schema Matching”, Proceedings of the 18th ICDE Conf., 2002.

[49] A. Doan, J. Madhaven, P. Domingos, A. Halevy, “Ontology Matching: A Machine Learning Approach”, Handbook on Ontologies in Information Systems, S. Staab and R. Studer (eds.), Springer-Verlag, 2004, pages 397-416.

[50] E. Rahm, P.A. Bernstein, “A survey of approaches to automatic schema matching”, The VLDB Journal 10 (2001), pp. 334–350.

[51] P. Shvaiko, J. Euzenat, “A Survey of Schema-Based Matching Approaches”, Journal on Data Semantics IV, LNCS 3730, pp. 146-171, 2005.

[52] Y. Kalfoglou, M. Schorlemmer, “Ontology mapping: the state of the art”, The Knowledge Engineering Review 18(1) pp. 1-31. 2003.

[53] M.J. Pazzani: “Representation of Electronic Mail Filtering Profiles: A User Study”, Proceedings of the 5th International Conference on Intelligent User Interfaces, pp. 202-206, New Orleans, USA, 2000.

[54] A.H.M. Cremers, J. Lindenberg and M.A. Neerincx, “Apples or Oranges?: A User-centred Framework for Co-operative User Profile Management”, 7th WWRF Meeting, Eindhoven, The Netherlands, December 2002.

[55] A. Waern and A. Rudstrom, “Can Readers Understand Their Profiles? A Study of Human Involvement in Reader Profiling”, Proceedings of the 34th Annual Hawaii International Conference on System Sciences, January 2001.

[56] S. Caokim, S. Sedillot, “Profiles Management for Personalised Services Provisioning”, 2nd European Conference on Universal Multiservice Networks, ECUMN 2002, Page(s):315 – 321, 8-10 April 2002.

[57] 3rd Generation Partnership Project (3GPP), “The Virtual Home Environment (Release 5)”, 3GPP TR 22.121 v5.3.1, http://www.3gpp.org/, June 2002.

[58] K. van der Sluijs, G.-J. Houben, “Towards a Generic User Model Component”, Workshop on Personalization on the Semantic Web (PerSWeb05) at the 10th International Conference on User Modeling (UM'2005), pp. 43-52, Edinburgh, Scotland, 25-26 July 2005.

[59] K. van der Sluijs, G.-J. Houben, “A Generic Component for Exchanging User Models between Web-based Systems”, International Journal of Continuing Engineering Education and Life-Long Learning, Vol. 16, Nos. 1/2, pp. 64-76, 2006.

[60] I. Horrocks, P.F. Patel-Schneider, H. Boley, S. Tabet, B. Grosof, M. Dean, “SWRL: A Semantic Web Rule Language Combining OWL and RuleML”, http://www.w3.org/Submission/SWRL/, W3C Member Submission, 21 May 2004.

[61] J. Groppe, W. Mueller, “Profile Management Technology for Smart Customizations in Private Home Applications”, Sixteenth International Workshop on Database and Expert Systems Applications, 2005.

[62] N. Houssos, A. Alonistioti, L. Merakos, M. Dillinger, M. Fahrmair, M. Schoenmakers, “Advanced Adaptability and Profile Management Framework for the Support of Flexible Mobile Service Provision”, IEEE Wireless Communications, Volume 10, Issue 4, Page(s):52 – 61, Aug. 2003.

[63] D. Morikawa, M. Honjo, A. Yamaguchi, M. Ohashi, “A Proposal of User Profile Management Framework for Context-Aware Service”, Symposium on Applications and the Internet Workshops, SAINT 2005 Workshops, Page(s):184 – 187, 2005.

[64] R. Etter, P. Dockhorn Costa, T. Broens, “A Rule-Based Approach Towards Context-Aware User Notification Services”, Proc. of the IEEE International Conference on Pervasive Services 2006, Lyon, France, June 2006.

[65] M. Román, C.K. Hess, R. Cerqueira, A. Ranganathan, R.H. Campbell, and K. Nahrstedt, “A Middleware Infrastructure to Enable Active Spaces”, In IEEE Pervasive Computing, pp. 74-83, Oct-Dec 2002.

[66] E. Chan, J. Bresler, J. Al-Muhtadi, R.H. Campbell, “Gaia Microserver: An Extendable Mobile Middleware Platform”, 3rd IEEE International Conference on Pervasive Computing and Communications (PerCom 2005), Kauai Island, HI, USA, 8-12 March, 2005.

[67] A. Ranganathan, R.H. Campbell, “A Middleware for Context-Aware Agents in Ubiquitous Computing Environments”, In ACM/IFIP/USENIX International Middleware Conference, 2003, Rio de Janeiro, Brazil, June 16-20, 2003.

[68] T. Gu, H. K. Pung, D. Q. Zhang, “Towards an OSGi-Based Infrastructure for Context-Aware Applications”, IEEE Pervasive Computing, Vol. 3, Issue 4, pages: 66 – 74, 2004.

[69] A. Ranganathan, S. Chetan, J. Al-Muhtadi, R.H. Campbell, M.D. Mickunas. “Olympus: A High-Level Programming Model for Pervasive Computing Environments”, In IEEE International Conference on Pervasive Computing and Communications (PerCom 2005), Kauai Island, Hawaii, March 8-12, 2005.

[70] D. Garlan, D.P. Siewiorek, A. Smailagic, P. Steenkiste, “Project Aura: Toward Distraction-Free Pervasive Computing”, IEEE Pervasive Computing, Vol.1, No.2, pp. 22-32, 2002.

[71] K. Henricksen, J. Indulska, T. McFadden, and S. Balasubramaniam, “Middleware for distributed context-aware systems”, In International Symposium on Distributed Objects and Applications (DOA), volume 3760 of Lecture Notes in Computer Science, pages 846-863, 2005.

[72] K. Henricksen, J. Indulska, “Personalising context-aware applications”, In OTM Workshop on Context-Aware Mobile Systems (CAMS), 2005.

[73] K. Henricksen, J. Indulska, “A Software Engineering Framework for Context-Aware Pervasive Computing”, Proceedings of the Second IEEE International Conference on Pervasive Computing and Communications (PerCom'04), Orlando, Florida, March 2004.

[74] J. Indulska, K. Henricksen, T. McFadden, P. Mascaro, “Towards a Common Context Model for Virtual Community Applications”, The 2nd International Conference On Smart Homes and Health Telematics (ICOST'2004), Singapore, September 2004.

[75] T. McFadden, K. Henricksen, J. Indulska, P. Mascaro, “Applying a Disciplined Approach to the Development of a Context-Aware Communication Application”, Proc. of the 3rd IEEE International Conference on Pervasive Computing and Communications (PerCom'05), Hawaii, March 2005.

[76] H. Chen, T. Finin, A. Joshi, “An Ontology for Context Aware Pervasive Computing Environments,” The Knowledge Engineering Review, Volume 18, Issue 3 (September 2003), pages 197-207, 2003.

[77] X. Wang, J.S. Dong, C.Y. Chin, S.R. Hettiarachchi, D. Zhang, “Semantic Space: An Infrastructure for Smart Spaces,” IEEE Pervasive Computing, Volume 3, Issue 3 (July 2004), pages 32-39, 2004.

[78] H. Chen, F. Perich, T. Finin, A. Joshi, “SOUPA: Standard Ontology for Ubiquitous and Pervasive Applications”, International Conference on Mobile and Ubiquitous Systems (MobiQuitous 2004), Boston, Massachusetts, USA, 2004.

[79] M. Luther, B. Mrohs, M. Wagner, S. Steglich, W. Kellerer, “Situational Reasoning – A Practical OWL Use Case,” International Symposium on Autonomous Decentralized Systems (ISADS), Chengdu, China, 2005.

[80] A. Harter, A. Hopper, “A Distributed Location System for the Active Office,”

IEEE Network, Volume 8, pp. 62-70, 1994.

[81] A. Harter, A. Hopper, P. Steggles, A. Ward, P. Webster, "The Anatomy of a Context-Aware Application," Proceedings of the International Conference on Mobile Computing and Networking, Seattle, USA, 1999.

[82] D. Beckett, B. McBride (Editors), "RDF/XML Syntax Specification (Revised)", http://www.w3.org/TR/rdf-syntax-grammar/, W3C Recommendation, 10 February 2004.

[83] T. Ottmann, P. Widmayer, "Algorithmen und Datenstrukturen," Spektrum Akademischer Verlag, ISBN 3827410290, 1996.

[84] A. Aho, J. Hopcraft and J. Ullman, "The Design and Analysis of Computer Algorithms," Addison-Wesley, ISBN 8131702057, 1974.

[85] H. Rajasekaran, P. Laitinen, G. Marton, R. Seidl, P. Weik, "Trust Framework and Service Delivery in SPICE", International Conference on Intelligence in Service Delivery Networks (ICIN 2007), Bordeaux, France, October 2007.

[86] H. van Kranenburg, M. Bargh, S. Iacob and A. Peddemors, "A Context Management Framework for Supporting Context Aware Distributed Applications", IEEE Communications Magazine 44[8], pp 67-74, 2006.

[87] C. Räck (Editor), "Specification of Pro-active Service Infrastructure for Attentive Services", SPICE project deliverable, http://www.ist-spice.org/nav/deliverables.htm, December 2007.

[88] N.F. Noy, D.L. McGuinness, "Ontology Development 101: A Guide to Creating Your First Ontology", Stanford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880, March 2001.

[89] Apache Software Organisation, "Apache Logging Services Project", http://logging.apache.org, September 2007.

[90] S. Gupta, "Pro Apache Log4j", Apress, ISBN 1590594991, June 2005.

[91] J. Grobler and D. Kourie, "Design of a High Resolution Soft Real-Time Timer under a Win32 Operating", Proceedings of SAICSIT 2005, White River, South Africa, 2005.

[92] Brian McBride, "An Introduction to RDF and the Jena RDF API", 2007. Available: http://jena.sourceforge.net/tutorial/RDF_API /index.html, 3 August 2008.

[93] H. Stöcker (Editor), "Taschenbuch der Physik", ISBN 3-8171-1556-3, Verlag Harry Deutsch, 1998.

[94] E. Prud'hommeaux, "W3C RDF Validation Service", 2007. Available: http://www.w3.org/RDF/Validator/, 3 August 2008.

[95] F. Baader and U. Sattler, "Tableau Algorithms for Description Logics",

International Conference TABLEAUX 2000, University of St Andrews, Scotland, July 2000.

[96] U. Hustadt, B. Motik and U. Sattler, “Reducing SHIQ-Description Logic to Disjunctive Datalog Programs”, Proc. of the 9th International Conference on Knowledge Representation and Reasoning (KR2004), pp 152-162, Whistler, Canada, June 2004.

[97] M.M. Shiaa and H. Demeter (Editors), “Final Reference Architecture”, SPICE project deliverable, http://www.ist-spice.org/nav/deliverables.htm, June 2008.

[98] A.E. Walsh (Editor), “UDDI, SOAP and WSDL – The Web Services Specification Reference Book”, ISBN 0-13-085726-2, Prentice Hall, 2002.

[99] E. Ceram, “Web Services Essentials”, ISBN 0596002246, O'Reilly, 2002.

[100] T. Berners-Lee, “Primer: Getting into RDF & Semantic Web using N3“,http://www.w3.org/2000/10/swap/Primer.html, August 2005.